Cognos® 8

FRAMEWORK MANAGER

USER GUIDE

COGNOS®

THE NEXT LEVEL OF PERFORMANCE™

Product Information

This document applies to Cognos® 8 Version 8.3 and may also apply to subsequent releases. To check for newer versions of this document, visit the Cognos Global Customer Services Web site (http://support.cognos.com).

Copyright

Copyright © 2007 Cognos Incorporated.

Portions of Cognos® software products are protected by one or more of the following U.S. Patents: 6,609,123 B1; 6,611,838 B1; 6,662,188 B1; 6,728,697 B2; 6,741,982 B2; 6,763,520 B1; 6,768,995 B2; 6,782,378 B2; 6,847,973 B2; 6,907,428 B2; 6,853,375 B2; 6,986,135 B2; 6,995,768 B2; 7,062,479 B2; 7,072,822 B2; 7,111,007 B2; 7,130,822 B1; 7,155,398 B2; 7,171,425 B2; 7,185,016 B1;7,213,199 B2.

Cognos and the Cognos logo are trademarks of Cognos Incorporated in the United States and/or other countries. All other names are trademarks or registered trademarks of their respective companies.

While every attempt has been made to ensure that the information in this document is accurate and complete, some typographical errors or technical inaccuracies may exist. Cognos does not accept responsibility for any kind of loss resulting from the use of information contained in this document.

This document shows the publication date. The information contained in this document is subject to change without notice. Any improvements or changes to either the product or the document will be documented in subsequent editions.

U.S. Government Restricted Rights. The software and accompanying materials are provided with Restricted Rights. Use, duplication, or disclosure by the Government is subject to the restrictions in subparagraph (C)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, or subparagraphs (C)(1) and (2) of the Commercial Computer Software - Restricted Rights at 48CFR52.227-19, as applicable. The Contractor is Cognos Corporation, 15 Wayside Road, Burlington, MA 01803.

This software/documentation contains proprietary information of Cognos Incorporated. All rights are reserved. Reverse engineering of this software is prohibited. No part of this software/documentation may be copied, photocopied, reproduced, stored in a retrieval system, transmitted in any form or by any means, or translated into another language without the prior written consent of Cognos Incorporated.

Table of Contents

Introduction 23

Chapter 1: What's New? 25

New Features in Version 8.3 25
Changed Features in Version 8.3 26
Deprecated Features in Version 8.3 26
Framework Manager Security Filters for SAP BW: Notice of Intent to Change the Default Setting 26
IQD Externalize Method 27
Removed Features in Version 8.3 27

Chapter 2: Getting Started with Framework Manager 29

Analyze the Problem 29
Building Cognos 8 Applications 30
Objects You Will Use 31
Create a Project 33
Open a Project 35
The Project Page 35
The Project Viewer 35
The Explorer Tab 38
The Diagram Tab 38
The Dimension Map Tab 39
The Properties Pane 39
The Tools Pane 41
Naming Conventions for Objects in a Project 43
Sample Models 44
The Great Outdoors Warehouse Model 45
The Great Outdoors Sales Model 46

Chapter 3: Importing Metadata from Data Sources 47

Data Sources 47
Data Source Security 47
Types of Data Source Connections 48
Cognos Cubes 48
Native Metadata 50
Microsoft SQL Server Data Sources 50
Microsoft Analysis Services Data Sources 51
ODBC Data Sources 53
IBM DB2 OLAP and Hyperion Essbase Data Sources 53
Attributes of a Dimension in IBM DB2 OLAP and Hyperion Essbase Data Sources 57
Composite Software 57
XML Data Sources 57
Working With Data Source Connections 58
Create a Data Source Connection 61

Recommendation - Use Network Paths for File-Based Data Sources 64
Importing Metadata 65
Import Metadata from a Relational Database 66
Import Metadata from a Cognos 8 Model 67
Import Metadata from an Architect Model or an Impromptu Catalog 68
Import Metadata from DecisionStream or Cognos 8 Data Manager 68
Import Metadata From a Third Party Metadata Source 72
Troubleshooting Metadata from Third Party Sources 76
Import Metadata Using XML as a Data Source 77
Import Objects with the Same Name 78

Chapter 4: Modeling Relational Metadata 79
Relationships 79
Cardinality 80
Modify a Relationship 83
Create a Complex Expression for a Relationship 84
Create a Relationship 84
Create a Relationship Shortcut 85
Detect and Generate Relationships 86
Query Subjects 87
Data Source Query Subjects 87
Model Query Subjects 89
Stored Procedure Query Subjects 90
Determinants 94
Create a Model Query Subject Based on Existing Objects 98
View Related Objects 99
Create a Query Set 99
Test a Query Subject or Query Set 103
Validate a Query Subject 106
Update Query Subjects 107
Convert a Query Subject into a Dimension 107
Convert a Model Query Subject into a Data Source Query Subject 108
Edit the SQL 108
Change the Type of SQL 109
Change How the SQL Is Generated 114
Dimensions 115
Normalized Data Sources 116
Create a Regular Dimension 117
Roles 123
Create a Measure Dimension 124
Convert a Measure into a Query Item 126
Scope Relationships 126
Create a Regular Dimension Based on Existing Objects 127
View Related Objects 127
Test a Dimension 128
Convert a Regular Dimension into a Query Subject 131
Multilingual Metadata 131
Setting Up a Multilingual Reporting Environment 132
Using a Macro to Model Multilingual Data 134

Add a Language to a Project *135*
Export a Translation Table *136*
Import a Translation Table *137*
Example - Create a Multilingual Project for Relational Metadata *137*
Query Items *139*
Modifying How Query Items Are Aggregated *141*
Format Query Items *148*
Define a Prompt Control *149*
Convert a Query Item into a Measure *153*
Adding Business Rules *153*
Create a Calculation *153*
Create a Filter *156*
Apply a Filter *158*
Example - Show the Currency Name for Each Country *160*
Create a Parameter Map *161*
Example - Specifying a Language Value for Relational Metadata *162*
Create a Session Parameter *163*
Using Parameters with Relational Data Source Query Subjects *164*
Creating Prompts with Query Macros *165*
Organizing the Model *175*
Create a Star Schema Group *176*
Use Shortcuts *180*
Create a Folder or Namespace *183*
Create a Query Item Folder *184*
Create a Measure Folder *185*
Analyze a Model *185*

Chapter 5: Working with SAP BW Metadata *191*

Import from an SAP BW Data Source *191*
Mapping SAP BW Objects to Framework Manager *197*
Dimensions *198*
Modify a Regular Dimension *199*
Roles *202*
Modify a Key Figures Dimension *205*
Conform SAP BW Dimensions *205*
View Related Objects *207*
Test a Dimension or Other Object *208*
Working with Model Query Subjects *211*
Query Items *213*
Modifying How Query Items Are Aggregated *216*
Format Query Items *219*
Define a Prompt Control *220*
Adding Business Rules *231*
Create a Calculation *231*
Create a Filter *233*
Apply a Filter *235*
Create a Parameter Map *237*
Create a Session Parameter *238*
Organizing the Model *240*

Use Shortcuts 240
Create a Folder or Namespace 242
Optimizing SAP BW Performance 243
Dimension Settings 243
Data Source Settings 245

Chapter 6: Publishing Packages 247
Verify a Model or Package 247
Create or Modify a Package 250
Package Security 252
Users, Groups, and Roles 253
Add Data Security 253
Add or Remove Object Security 255
Modify Package Security 258
Specify Languages 258
Externalizing Query Subjects and Dimensions 259
Publish a Package 261
Publish a Package Based on an OLAP Data Source 263
Publishing a Package by Running a Script 265
Update a Report to Use the Latest Version of a Package 265

Chapter 7: Managing the Project 267
Understanding the Metadata in Your Model 267
Explore a Package 267
View the Distribution of an Object in Packages 268
Create Model Documentation 268
Multiuser Modeling 269
Branching and Merging Projects 269
Segmenting and Linking Projects 276
Repository Control 280
Using External Repository Control 289
Administering the Metadata 289
Copy, Move, Rename, or Delete a Project 290
Analyze the Impact of Changes to a Package 292
Remap an Object to a New Source 295
Export Metadata 297
Project Reuse 298
Model Portability 299
Synchronize Projects 306
Query Behavior 309
Set Governors 310
Specify Where Aggregate Rollups are Processed 314
Improve Performance by Setting Query Processing Type 315
Improving Performance by Reusing Cached Data When Running a Report 315
Select Function Sets 317
Quality of Service 317
Control and Optimize How Queries Are Run 319

Chapter 8: Guidelines for Modeling Metadata 321
Understanding Cognos 8 Modeling Concepts 321

Relational Modeling Concepts 322
Model Design Considerations 330
Dimensional Modeling Concepts 335
Building the Relational Model 337
Defining the Relational Modeling Foundation 337
Defining the Dimensional Representation of the Model 344
Organizing the Model 347

Chapter 9: The SQL Generated by Cognos 8 351
Understanding Dimensional Queries 351
Single Fact Query 351
Multiple-fact, Multiple-grain Query on Conformed Dimensions 352
Modeling 1-n Relationships as 1-1 Relationships 355
Multiple-fact, Multiple-grain Query on Non-Conformed Dimensions 357
Resolving Ambiguously Identified Dimensions and Facts 361
Query Subjects That Represent a Level of Hierarchy 361
Resolving Queries That Should Not Have Been Split 362
Resolving Queries That Are Split in the Wrong Place 365

Chapter 10: Upgrading Models 371
Cognos ReportNet Model Upgrade 371
Verifying the Model in Cognos ReportNet 371
Opening and Upgrading the Model 372
Verifying and Repairing the Upgraded Cognos ReportNet Model 374
Cognos 8.1 or 8.2 Model Upgrade 378
Verifying the Model in Cognos 8.1 or 8.2 379
Opening and Upgrading the Cognos 8.1 or 8.2 Model 379
Verifying and Repairing the Upgraded Cognos 8.1 or 8.2 Model 379
Upgrading Segmented and Linked Projects 380

Appendix A: Troubleshooting 381
Measures Not Added After Initial Import in an Upgraded Model 381
Unable to View the Result Set of a Stored Procedure 381
Unable to Compare Two CLOBs in Oracle 381
An Out of Memory Error with ERWin Imported Metadata 382
Calculation Fails Testing 382
Framework Manager Cannot Access the Gateway URI 382
Object Names Appear in the Wrong Language 383
Full Outer Joins in Oracle Return Incorrect Results 383
Error When Testing Query Subjects in a Model Imported from Teradata 383
Error for Type-In SQL Query Subject 384
Function Name Not Recognized 384
QE-DEF-0259 Error 384
QE-DEF-0260 Parsing Error 385
Externalized Key Figures Dimension Retains Old Prompt Value 385
Older Cognos 8 Models Display Level Object Security 385
Exporting a Framework Manager Model to a CWM File Fails With Error *MILOG.TXT was not found* 386
Difference in SQL for Inner Joins After Upgrading to Cognos 8.3 386
Unable to Select a Command for a Query Item in a Shortcut 386

Publishing a Large Package Results in a CCLOutOfMemory Error 386

Appendix B: Using the Expression Editor 387

Searching for Values May Return Unexpected Results 388

Operators 389

(389

) 389

* 389

, 389

/ 389

|| 389

+ 390

- 390

< 390

<= 390

<> 390

= 390

> 391

-> 391

>= 391

and 391

auto 391

between 391

case 392

contains 392

currentMeasure 392

default 392

distinct 392

else 392

end 393

ends with 393

for 393

for all 393

for any 393

for report 393

if 394

in 394

in_range 394

is missing 394

is null 394

is not missing 394

is not null 395

like 395

lookup 395

not 395

or 395

prefilter 395

rows 396

starts with 396

then *396*
when *396*
Summaries *396*
aggregate *396*
average *397*
count *397*
maximum *397*
median *398*
minimum *398*
moving-average *398*
moving-total *399*
percentage *399*
percentile *400*
quantile *401*
quartile *402*
rank *402*
running-average *403*
running-count *404*
running-difference *404*
running-maximum *405*
running-minimum *405*
running-total *406*
standard-deviation *407*
standard-deviation-pop *407*
total *407*
variance *408*
variance-pop *408*
Member Summaries *408*
aggregate *408*
average *409*
count *409*
maximum *409*
median *409*
minimum *409*
percentage *409*
percentile *410*
quantile *410*
quartile *410*
rank *410*
standard-deviation *411*
total *411*
variance *411*
Constants *411*
date *411*
date-time *411*
time with time zone *411*
timestamp with time zone *412*
interval *412*
interval year *412*

interval month 412
interval year to month 412
interval day 412
interval hour 412
interval minute 412
interval second 412
interval day to hour 412
interval day to minute 412
interval day to second 412
interval hour to minute 413
interval hour to second 413
interval minute to second 413
null 413
number 413
string 413
time 413
Constructs 413
if then else 413
in_range 413
search case 414
simple case 414
Business Date/Time Functions 414
_add_days 414
_add_months 415
_add_years 415
_age 415
_day_of_week 416
_day_of_year 416
_days_between 416
_days_to_end_of_month 417
_first_of_month 417
_last_of_month 417
_make_timestamp 418
_months_between 418
_week_of_year 418
_years_between 418
_ymdint_between 419
Block Functions 419
_firstFromSet 419
_remainderSet 419
Macro Functions 420
+ 420
array 420
csv 420
dq 421
grep 421
join 421
prompt 422
promptmany 422

sb *423*
sq *423*
sort *423*
split *423*
substitute *424*
unique *424*
urlencode *425*
CSVIdentityName *425*
CSVIdentityNameList *425*
CAMPassport *426*
CAMIDList *426*
CAMIDListForType *426*
Common Functions *427*
nullif *427*
_format *427*
_round *427*
abs *427*
ancestor *428*
ancestors *428*
bottomCount *428*
bottomPercent *428*
bottomSum *428*
caption *429*
cast *429*
ceil *430*
ceiling *430*
char_length *430*
character_length *430*
children *431*
closingPeriod *431*
coalesce *431*
completeTuple *431*
cousin *431*
current_date *432*
current_time *432*
current_timestamp *432*
currentMember *432*
defaultMember *433*
descendants *433*
emptySet *433*
except *434*
exp *434*
extract *434*
filter *434*
firstChild *435*
firstSibling *435*
floor *435*
generate *435*
head *435*

hierarchize *436*
hierarchy *436*
intersect *436*
item *436*
lag *436*
lastChild *436*
lastPeriods *437*
lastSibling *437*
lead *437*
level *437*
levels *437*
linkMember *437*
ln *438*
localtime *438*
localtimestamp *438*
lower *438*
member *439*
members *439*
mod *439*
nestedSet *439*
nextMember *440*
octet_length *440*
openingPeriod *440*
order *440*
ordinal *441*
parallelPeriod *441*
parent *441*
periodsToDate *441*
position *441*
power *442*
prevMember *442*
roleValue *442*
rootMembers *443*
set *443*
siblings *443*
sqrt *443*
subset *443*
substring *444*
tail *444*
topCount *444*
topPercent *444*
topSum *444*
trim *445*
tuple *445*
union *445*
unique *445*
upper *446*
value *446*
DB2 *446*

ascii *446*
ceiling *446*
char *446*
chr *447*
concat *447*
date *447*
day *447*
dayname *447*
dayofweek *447*
dayofweek_iso *448*
dayofyear *448*
days *448*
dec *448*
decimal *448*
difference *449*
digits *449*
double *449*
event_mon_state *449*
float *449*
hex *449*
hour *450*
insert *450*
integer *450*
int *450*
julian_day *450*
lcase *451*
left *451*
length *451*
locate *451*
long_varchar *451*
ltrim *451*
microsecond *452*
midnight_seconds *452*
minute *452*
month *452*
monthname *452*
quarter *453*
radians *453*
repeat *453*
replace *453*
right *453*
round *453*
rtrim *454*
second *454*
sign *454*
smallint *454*
soundex *454*
space *454*
substr *455*

table_name 455
table_schema 455
time 455
timestamp 455
timestamp_iso 456
timestampdiff 456
to_char 456
translate 456
trunc 456
truncate 457
ucase 457
value 457
varchar 457
week 457
year 457
DB2 Cast 458
DB2 Math 459
DB2 Trigonometry 460
Informix 462
cardinality 462
char_length 462
concat 462
date 462
day 463
extend 463
hex 463
initcap 463
length 464
lpad 464
mdy 464
month 464
nvl 464
octet_length 464
replace 465
round 465
rpad 465
substr 465
to_char 465
to_date 466
trunc 466
weekday 466
year 466
Informix Math 466
Informix Trigonometry 467
MS Access 468
ascii 468
ceiling 468
chr 468
concat 468

curdate *469*
curtime *469*
dayname *469*
dayofmonth *469*
dayofweek *469*
dayofyear *469*
hour *470*
instr *470*
lcase *470*
left *470*
length *470*
locate *471*
ltrim *471*
minute *471*
month *471*
monthname *471*
now *471*
position *472*
quarter *472*
right *472*
round *472*
rtrim *472*
sign *472*
space *473*
substr *473*
substring *473*
truncate *473*
ucase *473*
week *473*
year *474*
MS Access Cast *474*
MS Access Math *475*
MS Access Trigonometry *475*
Oracle *476*
add_months *476*
ascii *476*
ceil *476*
char_length *476*
chr *477*
concat *477*
decode *477*
dump *477*
greatest *477*
initcap *477*
instr *478*
instrb *478*
least *478*
length *478*
lengthb *478*

lpad *478*
ltrim *479*
months_between *479*
new_time *479*
next_day *479*
nls_initcap *479*
nls_lower *480*
nls_upper *480*
nvl *480*
replace *480*
round *480*
rpad *481*
rtrim *481*
sign *481*
soundex *481*
substr *481*
substrb *482*
{sysdate} *482*
to_char *482*
to_date *482*
to_number *482*
translate *482*
trunc *483*
trunc *483*
{user} *483*
vsize *483*
Oracle Math *483*
Oracle Trigonometry *483*
Red Brick *485*
ceil *485*
concat *485*
{current_user} *485*
date *485*
dateadd *486*
datediff *486*
datename *486*
dec *486*
decimal *486*
decode *487*
float *487*
ifnull *487*
int *487*
integer *487*
length *488*
lengthb *488*
ltrim *488*
nullif *488*
positionb *488*
real *489*

round 489
rtrim 489
sign 489
string 489
substr 490
substrb 490
time 490
timestamp 490
timestamp 490
to_char 491
SQL Server 491
ascii 491
char 491
charindex 491
{current_user} 491
datalength 492
dateadd 492
datediff 492
datename 492
datepart 492
day 493
difference 493
getdate 493
left 493
ltrim 493
month 494
patindex 494
replace 494
replicate 494
reverse 494
right 495
round 495
rtrim 495
sign 495
soundex 495
space 495
str 496
stuff 496
year 496
SQL Server Cast 496
SQL Server Math 497
SQL Server Trigonometry 498
Teradata 499
account 499
add_months 499
bytes 500
case_n 500
char2hexint 500
characters 500

database 500
date 501
format 501
index 501
log 501
nullif 501
nullifzero 501
profile 502
random 502
role 502
session 502
soundex 502
substr 502
time 503
type 503
user 503
vargraphic 503
zeroifnull 503
Teradata Trigonometry 503
SAP BW 506
SAP BW Trigonometry 506
SAP BW Math 507
Sybase 507
ascii 507
char 508
charindex 508
datalength 508
dateadd 508
datediff 508
datename 509
datepart 509
day 509
difference 509
getdate 510
left 510
ltrim 510
month 510
patindex 510
rand 510
replicate 511
reverse 511
right 511
round 511
rtrim 511
soundex 511
space 512
str 512
stuff 512
substring 512

to_unichar *512*
uhighsurr *513*
ulowsurr *513*
uscalar *513*
year *513*
Sybase Math *514*
Sybase Trigonometry *514*
Report Functions *516*
_add_days *516*
_add_months *516*
_add_years *516*
_age *516*
_day_of_week *517*
_day_of_year *517*
_days_between *517*
_days_to_end_of_month *517*
_first_of_month *517*
_last_of_month *518*
_make_timestamp *518*
_months_between *518*
_week_of_year *518*
_years_between *518*
_ymdint_between *519*
abs *519*
AsOfDate *519*
AsOfTime *519*
BurstKey *519*
BurstRecipients *519*
ceiling *520*
CellValue *520*
character_length *520*
ColumnNumber *520*
CubeCreatedOn *520*
CubeCurrentPeriod *520*
CubeDataUpdatedOn *521*
CubeDefaultMeasure *521*
CubeDescription *521*
CubeIsOptimized *521*
CubeName *521*
CubeSchemaUpdatedOn *521*
exp *522*
extract *522*
floor *522*
GetLocale *522*
HorizontalPageCount *522*
HorizontalPageNumber *522*
InScope *523*
IsBursting *523*
IsCrosstabColumnNodeMember *523*

IsCrosstabRowNodeMember *523*
IsFirstColumn *523*
IsInnerMostCrosstabColumnNodeMember *523*
IsInnerMostCrosstabRowNodeMember *524*
IsLastColumn *524*
IsLastInnerMostCrosstabColumnNodeMember *524*
IsLastInnerMostCrosstabRowNodeMember *524*
IsOuterMostCrosstabColumnNodeMember *524*
IsOuterMostCrosstabRowNodeMember *524*
IsPageCountAvailable *524*
ln *525*
Locale *525*
lower *525*
mapNumberToLetter *525*
mod *525*
ModelPath *525*
Now *526*
nullif *526*
octet_length *526*
PageCount *526*
PageName *526*
PageNumber *526*
ParamCount *527*
ParamDisplayValue *527*
ParamName *527*
ParamNames *527*
ParamValue *527*
position *527*
power *528*
ReportAuthorLocale *528*
ReportCreateDate *528*
ReportDate *528*
ReportDescription *528*
ReportID *528*
ReportLocale *529*
ReportName *529*
ReportOption *529*
ReportOutput *529*
ReportPath *529*
ReportProductLocale *529*
ReportSaveDate *530*
round *530*
RowNumber *530*
ServerLocale *530*
ServerName *530*
sqrt *530*
substring *531*
TOCHeadingCount *531*
Today *531*

trim *531*
upper *531*
URLEncode *531*
Data Type Casting Functions *532*

Appendix C: Data Formatting Reference *537*
Data Formatting Properties *537*
"Not Applicable" Characters *537*
Any Error Characters *537*
Calendar Type *537*
Clock *537*
Currency *538*
Currency Display *538*
Currency Symbol *538*
Currency Symbol Position *538*
Date Ordering *538*
Date Separator *538*
Date Style *538*
Decimal Separator *538*
Display AM / PM Symbols *539*
Display As Exponent *539*
Display Days *539*
Display Eras *539*
Display Hours *539*
Display Milliseconds *539*
Display Minutes *539*
Display Months *539*
Display Months *539*
Display Seconds *540*
Display Time Zone *540*
Display Weekdays *540*
Display Years *540*
Display Years *540*
Divide By Zero Characters *540*
Exponent Symbol *540*
Group Size (digits) *540*
International Currency Symbol *540*
Mantissa (digits) *541*
Maximum No. of Digits *541*
Minimum No. of Digits *541*
Missing Value Characters *541*
Negative Pattern *541*
Negative Sign Position *541*
Negative Sign Symbol *541*
No. of Decimal Places *541*
Numeric Overflow Characters *542*
Padding Character *542*
Pattern *542*
Percentage Symbol *542*

Percent Scale (integer) *542*
Scale *542*
Secondary Group Size (digits) *542*
Security Error Characters *542*
Thousands Separator *543*
Time Separator *543*
Time Style *543*
Time Unit *543*
Use Thousands Separator *543*
Zero Value Characters *543*

Appendix D: Using Patterns to Format Data *545*
Pattern Guidelines *545*
Date and Time Symbols *546*
Decimal Format Symbols *555*

Appendix E: Guidelines for Working with SAP BW Data in Other Applications *557*
Working with SAP BW Data in Transformer 8.x and 7.x *557*
SAP BW Query Requirements *559*
Framework Manager Considerations *560*
Building PowerCubes from SAP BW Data *561*
Work with SAP BW Data in Cognos 8 Planning *563*

Appendix F: Reserved Words *565*

Appendix G: XML Data Types *569*

Glossary *571*

Index *575*

Introduction

Framework Manager is a metadata modeling tool. A model is a business presentation of the information in one or more data sources. When you add security and multilingual capabilities to this business presentation, one model can serve the needs of many groups of users around the globe.

This document includes the procedures, examples, notes, tips, and other background information to help you prepare a model for reporting and deploying a package.

Audience

This document is intended to help data modelers use Framework Manager. Before using Framework Manager, you should understand data modeling and how to write queries.

Related Documentation

Our documentation includes user guides, getting started guides, new features guides, readmes, and other materials to meet the needs of our varied audience. The following documents contain related information and may be referred to in this document.

You can access the following documents on the Cognos Global Customer Services Web site (http://support.cognos.com). Logon credentials are available either from your administrator or by request from support.america@cognos.com.

Document	Description
Analysis Studio *User Guide*	Exploring, analyzing, and comparing dimensional data
Cognos 8 Administration and Security *Guide*	Managing servers, security, reports, and portal services; setting up Cognos samples; troubleshooting; and customizing Cognos 8
Cognos 8 Installation and Configuration *Guide*	Installing, upgrading, configuring, and testing Cognos 8, changing application servers, and setting up samples
Cognos 8 Migration Tools *User Guide*	Moving metadata and applications from Cognos Series 7 to Cognos 8
Event Studio *User Guide*	Creating and managing agents that monitor data and perform tasks when the data meets predefined thresholds
Query Studio *User Guide*	Creating self-service business intelligence reports

Document	Description
Report Studio Professional Authoring *User Guide*	Authoring reports that analyze corporate data according to specific needs
Report Studio Express Authoring *User Guide*	Authoring financial reports that analyze corporate data according to specific needs

Finding Information

To find the most current product documentation, including all localized documentation, access the Cognos Global Customer Services Web site (http://support.cognos.com). Click the **Documentation** link to access documentation guides. Click the **Knowledge Base** link to access all documentation, technical papers, and multimedia materials.

Product documentation is available in online help from the **Help** menu or button in Cognos products. You can also download documentation in PDF format from the Cognos Global Customer Services Web site.

You can also read PDF versions of the product readme files and installation guides directly from Cognos product CDs.

Using Quick Tours

Quick tours are short online tutorials that illustrate key features in Cognos product components. To view a quick tour, start Cognos Connection and click the **Quick Tour** link in the lower-right corner of the Welcome page.

Getting Help

For more information about using this product or for technical assistance, visit the Cognos Global Customer Services Web site (http://support.cognos.com). This site provides product information, services, user forums, and a knowledge base of documentation and multimedia materials. To create a case, contact a support person, or to provide feedback, click the **Contact Us** link. For information about education and training, click the **Training** link.

Chapter 1: What's New?

This section contains a list of new, changed, deprecated, and removed features for this release. It will help you plan your upgrade and application deployment strategies and the training requirements for your users.

For information about upgrading, see the *Installation and Configuration Guide* and "Upgrading Models" (p. 371).

For information about new features for this release, see the *New Features Guide*.

To review an up-to-date list of environments supported by Cognos products, such as operating systems, patches, browsers, Web servers, directory servers, database servers, and application servers, visit the Cognos Global Customer Services Web site (http://support.cognos.com).

New Features in Version 8.3

Listed below are new features since the last release. Links to directly-related topics are included.

- You can analyze the metadata in a model by using the **Model Advisor**, an automated tool that applies current modeling guidelines and identifies inconsistencies and areas in your model that you need to examine. For more information, see "Analyze a Model" (p. 185).
- You can specify the behavior of shortcuts by using the **Shortcut Processing** governor and the **Treat As** property. For more information, see "Set Governors" (p. 310) and "Use Shortcuts" (p. 180).
- You can minimize the effect of model changes and data source changes by using the **Remap To New Source** command. This command remaps higher-level model objects so that they continue to run and return correct data. You can remap query items, measures, calculations, and filters. You can remap individual objects manually or you can remap multiple objects at the same time. For more information, see "Remap an Object to a New Source" (p. 295).
- You can understand the full impact of potential changes in the model by using the **Show Object Dependencies** command to find the objects that depend on an object. For more information, see "Show Object Dependencies" (p. 294).
- You can branch and distribute a model for development and later collect and merge the changes made by multiple modelers by using the **Branch to** and **Merge from** commands. For more information, see "Branching and Merging Projects" (p. 269).
- You can import SAP BW queries that contain dual structures and use the structures in Cognos 8 queries to control the amount and order of information that your users see by using the **SAP BW Dual Structures Support** check box. For more information, see "SAP BW Structures" (p. 195).

Changed Features in Version 8.3

Listed below are changes to features since the last release. Links to directly-related topics are included.

- You can now publish packages from Framework Manager into any folder in Cognos Connection.
- With the appropriate capabilities and permissions, your users can now add PowerCubes directly from Cognos Connection without having to access Framework Manager.
- Before publishing a package and running reports, you can now see more information about how the changes you make to a model will affect the package and the reports that use it by using the **Analyze Publish Impact** command. For more information, see "Analyze the Impact of Changes to a Package" (p. 292).
- The **Verify Model** dialog box is improved. You can now select the level of verification that you want to perform. You can sort and group the reported items by type and by severity of the message. For more information, see "Verify a Model or Package" (p. 247).
- Diagrams in the **Diagram** tab are improved for performance, stability, and usability. You can now control the settings for diagrams and the **Context Explorer**. For more information, see "The Diagram Tab" (p. 38) and "Change the Settings for Diagrams" (p. 38).
- Search and replace is improved so that you can do bulk searching and replacing. Use the **Search** tab to quickly find objects by applying different search criteria, such as the location, class, condition, or property. Text properties such as Name, Description, and Screen Tip, are writable so you can easily replace multiple values. For more information, see "The Search Tab" (p. 41) and "Replacing Multiple Property Values" (p. 40).

Deprecated Features in Version 8.3

A deprecated feature is one that is being replaced by a newer version or a better implementation. Cognos intends to discontinue the use of the feature and provides recommendations for adapting to this change over multiple releases.

Listed below are deprecated features.

Framework Manager Security Filters for SAP BW: Notice of Intent to Change the Default Setting

In all shipped versions of Cognos ReportNet and Cognos 8 including Cognos 8.2, the following behavior has been enabled by default.

Multiple security filters defined within Framework Manager on metadata imported from SAP BW sources are combined using 'AND' logic, effectively an intersection of a particular user's permissions. This behavior is contradictory to corresponding behavior on relational data sources where similar filters are combined using 'IN' and 'OR' (union) logic to facilitate cases where users belong to one or more group and require a union of their permissions. The current default behavior for SAP BW

datasources has been determined to be a product defect and will be changed in the Cognos 8.3 release to align with the behavior of relational data sources. In the Cognos 8.1 Mr2 and Cognos 8.2 releases, it will be possible to get the union of filters behavior by modifying the following switches in the qfs_config.xml configuration file. Under `<provider name="OlapQueryProvider" libraryName="oqp">`, add the following new `<parameter>` element:

```
<parameter name="ORingSecurityFiltersWhenUserBelongsToMultipleGroups"
value="true"/>
```

Effect

When multiple security filters are defined in the Data Security setting in Framework Manager for a query subject, if a user belongs to more than one user group associated with these filters, the effect of this switch is to perform a union of these filters instead of an intersection. For example, if a user Joe belongs to a corporate group allowing him to see data for Asia, Europe, and America, and also belongs to a regional group allowing him only to see data for Europe, the effect of the switch will be to let Joe see data for Asia, Europe, and America (as compared to just Europe) when logging on Cognos 8 and authoring or running a report. Note that security filters that are based on other security filters in the Data Security settings for a query subject continue to be intersected, just as before, independent of whether or not the new switch is activated.

As of Cognos 8.3, this behavior will become the new default for SAP BW metadata in Framework Manager. There will be no change to the behavior of security filters applied on relational sources. The switch will continue to be available and can be set to false by your administrator if required to maintain existing application behavior.

IQD Externalize Method

IQD's generated by Framework Manager will continue to be supported in this Cognos 8 release, but will not be enhanced. Cognos will deprecate support for IQD's generated by Framework Manager in the next major release of Cognos 8. Cognos 8 Transformer will continue to support IQD's generated by Impromptu in the next major release.

Removed Features in Version 8.3

The functionality listed below has been removed from version 8.3.

- The **Embedded** externalize method has been removed from the current release. There is no impact on product functionality.

Chapter 2: Getting Started with Framework Manager

Framework Manager is a metadata modeling tool that drives query generation for Cognos 8. A model is a collection of metadata that includes physical information and business information for one or more data sources. Cognos 8 enables performance management on normalized and denormalized relational data sources and a variety of OLAP data sources. When you add security and multilingual capabilities, one model can serve the reporting, ad hoc querying, and analysis needs of many groups of users around the globe.

Before doing anything in Framework Manager, we recommend that you thoroughly understand the reporting problem that you want to solve (p. 29).

To get started with Framework Manager, we recommend that you do the following:

- Learn about the objects you will use (p. 31).
- Create (p. 33) or open (p. 35) a project.
- Explore the panes in Framework Manager (p. 35).
- Explore the sample models included with Framework Manager (p. 44).

Analyze the Problem

Before you start, you must understand the reporting problem that you are trying to solve and what data is available to solve it.

If you cannot address the following questions, talk to your users about their reporting requirements:

- Do you and your users agree on the reporting requirements?

  Issues to resolve can include multilingualism, performance, security, and how to organize and combine query items and filters.

- Does the data source contain the data and metadata that you need?

  Without metadata such as primary keys, indexes, and foreign keys, your reports may take too long to run, or may produce incorrect results. If the data source does not contain the data and metadata that you need, will it be changed, or will you work around it?

- Does the same data exist in more than one source?

  If so, choose the data source that most closely fits your reporting requirements. If a data warehouse is available, it is typically a better choice than an operational database. A data warehouse based on a star schema is ideal. If this does not exist, and you expect that your reporting application will be heavily used, consider arranging for one to be created.

- Which data source tables are the fact tables, which are the dimensions, and which are both fact table and dimension?
- What are the keys and attributes of each dimension?
- Which relationships are required?
- Are there multiple relationship paths between tables?

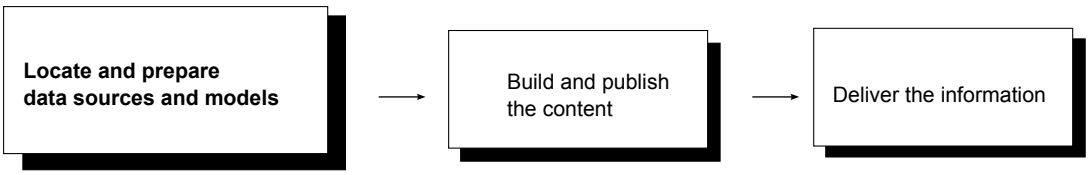
  If so, what does each path represent? You must define the preferred path for each.

Then you should review the names of data sources, tables, and columns in your data source to ensure that you are not using names reserved by Cognos 8. If you must use a reserved word, enclose the word in quotes in the SQL specification. For example, `select Orderdate, "Timezone"`. For more information, see "Reserved Words" (p. 565).

Building Cognos 8 Applications

The lifetime of a Cognos 8 business intelligence application can be months, or years. During that time, data may change, and new requirements appear, so authors must modify existing content and develop new content, and administrators must update models and data sources as the underlying data changes. For more information about using data sources, see the *Administration and Security Guide* and the Framework Manager *User Guide*.

In a working application, the technical and security infrastructure and the portal are in place, as well as processes for change management, data control, and so on. For information about the workflow associated with creating Cognos 8 content, see the *Architecture and Deployment Guide*. For more information, see the Cognos Solutions Implementation Methodology toolkit, which includes implementation roadmaps and supporting documents. The Cognos SIM toolkit is available on the Cognos Global Services Web site, http://support.cognos.com/en/consulting/csim/index.html.



- locate and prepare data sources and models

  Cognos 8 can report from a wide variety of data sources, both relational and dimensional. Database connections are created in the web administration interface, and are used for modeling, for authoring and for running the application.

  To use data for authoring and viewing, the business intelligence studios need a subset of a model of the metadata (called a package). The metadata may need extensive modeling in Framework Manager.

- build and publish the content

  Reports, scorecards, and so on are created in the business intelligence studios of Cognos 8. Which studio you use depends on the content, lifespan and audience of the report, and whether the data is modeled dimensionally or relationally. For example, self-service reporting and

analysis are done through Query Studio and Analysis Studio, and scheduled reports are created in Report Studio. Report Studio reports and scorecards are usually prepared for a wider audience, published to Cognos Connection or another portal, and scheduled there for bursting, distribution, and so on. You can also use Report Studio to prepare templates for self-service reporting.

- deliver and view the information

  You deliver content from the Cognos portal or other supported portals, and view information that has been saved to portals, or delivered by other mechanisms. You can also run reports, analyses, scorecards, and so on from within the business intelligence studio in which they were created.

  For information about tuning and performance, see the *Administration and Security Guide* and the Cognos Global Services Web site, http://support.cognos.com.

Objects You Will Use

When you work in Framework Manager, you work with a number of objects that are contained in a project.

Projects

A project contains a model, namespaces, packages, data sources, and related information for maintaining and sharing model information. A single project can span many data sources or tables.

A Framework Manager project appears as a folder that contains a project file (.cpf) and the specific .xml files that define the project. The files in a project folder are unique to each project. The project and its associated files are contained in a project folder.

We do not recommend adding secondary files to the project folder because they may be affected by actions such as move, rename, and delete commands on the **Manage Projects** menu. If you decide to add secondary files to the project folders, the files are added with absolute paths. If they are moved from the original location, they must be retargeted.

These are the contents of a project folder.

File name	Description
<project name>.cpf	The Framework Manager project file, which references the .xsd and .xml files used to define a project.
IDLog.xml	The file used to track objects for models that use branching and merging.
log.xml	A list of all modifications made to the model.
model.xml	The actual model data created by Framework Manager users.
preferences.xml	The preferences for Framework Manager projects.

File name	Description
persistent.txt	The stored diagram information, such as the diagram layout, notation, font, and color.
settings.ini	Temporary settings for the project that are created after you use the navigation window for the first time; these settings are not the same as the preferences.
repository.xml	The logged version history for each project or segment that was added to a repository; this file exists only if you added projects to a repository.

Models

A model is the set of related dimensions, query subjects, and other objects required for one or more related reporting applications.

The Framework Manager model is a metadata layer that adds value to a data source in several ways. Most importantly, it provides a business view of the information in the source data to simplify building reports, analyses, and queries. The business view can

- organize items in folders that represent business areas for reporting
- format items using numeric, currency, date, time, and other formats
- present multilingual folder and item names, descriptions, tips, and data so that users can operate in their language of choice
- automate the generation of SQL queries sent to the relational data source
- specify default prompting

  This can include having Cognos 8 prompt the user using a descriptive name while actually filtering on a code or key value for improved query performance.

In particular, you can modify the Framework Manager model to ensure that queries sent to the data source are efficient, well formed, and secure. You can specify the rules governing query generation, restrict user access to specific rows or columns of data, and model data relationships to hide the complexity of data from your users.

Namespaces

A namespace uniquely identifies query items, dimensions, query subjects, and other objects. You import different databases into separate namespaces to avoid duplicate names.

Packages

A package is a subset of the dimensions, query subjects, and other objects defined in the project. A package is what is actually published to the Cognos 8 server, and it is used to create reports, analyses, and ad hoc queries.

Dimensions

A dimension is a broad grouping of data about a major aspect of a business, such as products, dates, or markets.

The types of dimensions that you can work with in Framework Manager are regular dimensions and measure dimensions. In SAP BW, measure dimensions are called key figures.

Query Subjects

A query subject is a set of query items that have an inherent relationship.

In most cases, query subjects behave like tables. Query subjects produce the same set of rows regardless of which columns were queried.

There are different types of query subjects:

- data source

  Data source query subjects directly reference data in a single data source. Framework Manager automatically creates a relational data source query subject for each table and view that you import into your model.

- model

  Model query subjects are not generated directly from a data source but are based on query items in other query subjects or dimensions, including other model query subjects. By using model query subjects, you can create a more abstract, business-oriented view of a data source.

- stored procedure

  Stored procedure query subjects are generated when you import a procedure from a relational data source. Framework Manager supports only user-defined stored procedures. System stored procedures are not supported.

Query Items

A query item is the smallest piece of the model that can be placed in a report. It represents a single characteristic of something, such as the date that a product was introduced.

Query items are contained in query subjects or dimensions. For example, a query subject that references an entire table contains query items that represent each column in the table.

For your users, query items are the most important objects for creating reports. They use query item properties of query items to build their reports.

Create a Project

In Framework Manager, you work in the context of a project. The Framework Manager project contains objects that you organize for your users according to the business model and business rules of your organization. You view these objects in the project page (p. 35).

Before you can import metadata, you must create a project.

For information about using the contents of a repository to create a new project, see "Create a Local Project From a Repository" (p. 284).

For information about creating a project segment, see "Create a Segment" (p. 278).

Steps

1. From the **Welcome** page, click **Create a new project**.

   Tip: If you are in Framework Manager, click **New** from the **File** menu.

2. In the **New Project** page, specify a name and location for the project, and click **OK**.

3. In the **Select Language** page, click the design language for the project.

   You cannot change the language you select after you click **OK**, but you can add others. For more information, see "Add a Language to a Project" (p. 135).

   Note: If an SAP BW server does not support the selected language, it uses the content locale mapping in Cognos Configuration. If a mapping is not defined, Framework Manager uses the default language of the SAP BW server.

4. Click **OK** to select the design language.

   The **Metadata Wizard** appears.

5. Choose whether to import your metadata now or later:

   - To import now, select the import source and click **Next**.
   - To delay importing metadata, click **Cancel**.

6. If you chose to import the metadata now, follow the instructions in the **Metadata Wizard**:

   - Select a data source connection and click **Next**.

     If the data source connection you want is not listed, you must first create it (p. 61).

   - Select the check boxes for the objects you want to import.
   - Specify how the import should handle duplicate object names. Choose whether to import and create a unique name. If you choose to create a unique name, the imported object appears with a number. For example, you see QuerySubject and QuerySubject1 in your project.
   - If you want to import system objects, select the **Show System Objects** check box, and then select the system objects that you want to import.
   - Specify the criteria to use to create relationships and click **Import**.

     For more information, see "Relationships" (p. 79).

   You see a list of objects that could not be imported and a count of objects that were imported.

7. Click **Finish**.

   Save the project file (.cpf) and all related files in one folder. When you save a project with a different name or format, ensure that you save the project in a separate folder.

Open a Project

You must open a project before you can import metadata or make changes to existing metadata.

If the project was created using a model schema that is older than the currently supported version, you are prompted to upgrade the model.

If your model is checked into the repository, you cannot upgrade it. Manually check the model out of the source control system and then open it in the new version of Framework Manager.

If you upgrade a segmented model, you must open and upgrade each segment individually. After upgrading each segment, you can then upgrade the top level, or master, project.

Steps

1. From the **Welcome** page, click **Open a project**.

   Tip: If you are in Framework Manager, click **Open** from the **File** menu.

2. Browse to locate the project folder and click the .cpf file.

3. Click **OK**.

The Project Page

After you create (p. 33) or open (p. 35) a project, the project page appears. The project page is where you design, package, and publish project metadata. This page contains several panes and views that you can use to view and modify the objects in a project.

The Project Viewer

The **Project Viewer** shows the objects in a project in a hierarchical view. You can use the **Project Viewer** to view, modify, and create objects.

Relationships are shown in both the **Diagram** and **Explorer** tabs. Imported functions are shown in the **Explorer** tab.

The **Project Viewer** uses the following icons to represent objects and states. A project may use some or all of the icons.

Icon	Object
	Project
	Root namespace or any other namespace in the project
	Data source folder

Icon	Object
	Data source
	Parameter map folder
	Parameter map
	Packages folder
	Package
	Published package
	Dimension
	Hierarchy
	Level in a hierarchy
	Measure dimension
	Measure
	Query subject
	Query subject based on multidimensional data
	Query subject that contains query items whose **Usage** property is set to **Fact**
	Query item
	Calculation whose **Usage** property is set to **Attribute**
	Calculation whose **Usage** property is set to **Identifier**

Icon	Object
	Embedded calculation
	Calculation whose **Usage** property set to **Fact**
	Query item whose **Usage** property is set to **Identifier**
	Query item whose **Usage** property is set to **Fact**
	Query item that is located under a shortcut query subject
	Calculation
	Filter
	Relationship
	Semi-additive measure
	The object is checked into the repository. This icon appears over other icons.
	The object is checked out of the repository. This icon appears over other icons.
	The object is checked out of the repository by a different user. This icon appears over other icons.
	Linked segment or project that was updated. This icon appears over other icons.
	Linked object. This icon appears over other icons.
	Shortcut. This icon appears over other icons.
	Invalid object. This icon appears over other icons.

The Explorer Tab

The **Explorer** tab shows the contents of a project, similar to any file system. Arrange objects by name, class, or description. If you have a large number of objects in a project, it may be easier to locate them in the **Explorer** tab.

You can use the **Explorer** tab to view, create, and modify objects and relationships. You can also create folders and namespaces to group objects.

The Diagram Tab

Use the **Diagram** tab to show the relationships between objects in a project. Relationships between objects are shown as lines with cardinality notation (p. 81). You can expand objects and namespaces to show the object hierarchy and the relationships between objects.

In the **Diagram** tab, you can do any of the following:

- view, create, and modify objects and relationships.
- create folders and namespaces to group objects.
- change the settings for the diagrams (p. 38)
- change the layout of objects to either star layout or standard layout by clicking **Auto Layout** from the **Diagram** menu
- focus on an object by clicking **Set Focal Point** from the **Diagram** menu.
- find an object by right-clicking the object in the **Project Viewer** and clicking **Locate in Diagram**.
- zoom in or out by clicking **Zoom** from the **Diagram** menu.
- expand or collapse all objects from the **Diagram** menu.
- focus on the dimensions in the model by clicking **Dimensions view** from the **Diagram** menu.
- save the diagram for printing by clicking **Screen capture** from the **Diagram** menu and specifying the name of the picture.

Change the Settings for Diagrams

You can change one or more settings for diagrams.

Steps

1. From the **Diagram** menu, click **Diagram settings**.
2. Set the level of detail.

   You can select query items, relationships, scope relationships, cardinality, and descriptions.
3. Select the type of notation.

   You can use Merise or Crowsfeet notation. By default, Framework Manager uses Merise notation. Merise notation marks each end of the relationship with the minimum and maximum

cardinality of that end. You can also use Crowsfeet notation, which provides a pictorial representation of the relationship.

4. To align objects more easily, turn the **Snap options** on.
5. To have a network of evenly spaced lines in the background, select the **Display grid** check box.
6. Select the font and color for text.
7. If you are working in the **Context Explorer**, under **Options for Context Explorer**, specify showing objects with or without their related objects.

   If you change this option, it is applied the next time you open the **Context Explorer**.
8. Click **OK**.

The Dimension Map Tab

You can use the **Dimension Map** tab to view, create, and modify hierarchies and levels for the dimension you selected in the **Project Viewer**. You can also view and modify scope relationships.

The **Measures** and **Attributes** tabs appear after you click the **Dimension Map** tab. Use the **Measures** tab to view or modify all the measures and scope relationships that are available in the model. Use the **Attributes** tab to view or modify the role of the selected query item.

You can also embed calculations in the query item.

Tip: The best way to view SAP BW metadata is in the star layout. From the **Diagram** menu, click **Auto Layout Diagram**, and then select **Star**.

The Properties Pane

The **Properties** pane shows the properties of the objects that you last selected in the **Project Viewer**, **Explorer** tab, **Diagram** tab, **Dimension Map** tab, **Dependencies** tab, **Search** pane, or **Summary** pane. Object properties are set during import, and some property values can be modified during modeling. You can use the **Properties** pane to add, modify, or delete the properties of objects.

You can modify the properties for multiple objects at one time. If you select more than one object, Framework Manager shows only the properties that are common to all the objects. You can

- sort property values by double-clicking the property heading

  An arrow appears to indicate the direction in which values are sorted. You can toggle between ascending and descending order.
- filter property values by clicking the arrow to the right of the property heading

  You can either click a value, or click **Custom** to define the criteria for the rows that you want to view.
- apply a property value to multiple objects by clicking the arrow next to the property and dragging the highlighted area over the properties to which you want to apply that value
- resize the width of the rows and columns by right-clicking the object name in the property pane

If you need more room, you can resize the **Properties** pane or move it so that it floats in the window. For example, if you have more than one monitor, you can then move the **Properties** pane to another monitor.

Replacing Multiple Property Values

You can replace multiple values for each text string property.

When you replace multiple property values, Framework Manager automatically updates elements that are dependent on the values you replaced. If any dependencies cannot be updated, such as data source and parameter map names, you must validate the model manually.

You can match either the entire property value, or part of the value. By matching part of the property value, you can perform multiple replacements on a single text property. For example, if you search for “nation” and replace it with “country”, the property value “This nation and that nation” is replaced with “This country and that country”. However, "national" will be replaced with "countryal".

Tip: Before replacing property values, enlarge the **Properties** pane and widen the column of the property whose value you want to replace.

Steps

1. In the **Project Viewer** window, select multiple objects.

   Tip: To reduce the set of properties to search for before performing the replace, filter the properties first. On the **Properties** tab, right-click the column heading you want to filter on and click **Set Autofilter**.

2. Choose one of the following:

   - If your model contains multiple languages, click the **Language** tab.
   - If your model contains only one language, click the **Properties** tab.

3. Right-click the column heading for the property whose values you want to replace, and click **Bulk Replace**.

4. In the **Search for** and **Replace with** boxes, type the search and replace text strings.

   Tip: To replace empty properties, leave the **Search for** box blank.

5. To perform a case-sensitive search, select the **Match case** check box.

6. To search for the complete property text, select the **Match entire cell contents** check box.

   Tip: To replace all property fields, regardless of the text they contain, type one asterisk in the **Search for** box and select the **Match entire cell contents** check box.

7. Do one or more of the following:

   - Click **Replace** to replace the currently selected object and find the next match.
   - Click **Next** to select the next matching object.

- Click **Replace All** to replace all matching properties, starting from the first matching property in the list.
- Click **Close** if you are finished replacing.

As the changes to a property are made, the results appear on the tab.

When the bulk replace reaches the last object in the list, a message appears showing the number of replacements.

The Tools Pane

The **Tools** pane contains the **Summary** tab, the **Search** tab, and the **Dependencies** tab.

You can change the **Tools** pane into a separate window. To do this, click the title bar of the pane and drag it in any direction until the outline of pane becomes a thick line. You can now drag the **Tools** window to any location on your desktop. To change the window back to a pane inside the Framework Manager window, drag it to one of the Framework Manager borders until the window outline becomes a thin line.

The Summary Tab

The **Summary** tab shows the language, statistics, and tasks available for the selected object in the **Project Viewer**.

The **Project** section shows the design language and the active language. You can change the active language.

The **Statistics** section shows the number of objects, by class, located in the currently selected object. If the selected object contains a folder, the contents of the folder are included in the number count. Selected objects include projects, namespaces, and folders. The default selected object is the project.

The **Tasks** section shows actions that are applicable to the currently selected object, based on the object class. If you select a folder, actions for the folder are listed. If you select an object in that folder, the list includes actions for both the object and the folder.

For more information about the object classes, see the *c8_location*\templates\bmt\CR1Model\BMTModelSpecification.xsd file.

The Search Tab

When you are working with a large project, it can be difficult to locate the objects that you need to complete a task. Use the **Search** tab to quickly find objects by applying different search criteria, such as the location, the class, a condition, or a property.

If your model contains multiple languages, ensure that you specify the **Active language** value in the **Define Languages** dialog box before searching for objects.

Steps

1. In the **Tools** pane, click the **Search** tab.

   Tips:

- If the **Tools** pane is not visible, from the **View** menu, click **Tools**. You can drag the **Tools** pane to the bottom of the Framework Manager window and resize it to have a better view of the search results.
- To reduce the set of search properties, filter the properties first. On the **Properties** tab, right-click the column header you want to filter on, and click **Set Autofilter**.

2. In the **Search String** box, type the text that you want to find.

   You can use uppercase, lowercase, or mixed case strings. The search is not case sensitive. Valid wildcard characters are "*" and "?".

3. Click the **double up arrow** button to show the search criteria boxes.

4. In the **Condition** list, select a condition to apply to the search string.

   The **Condition** box determines how the **Search string** value is matched with text in the model. It contains a list of possible search conditions. If you want to search using wildcard characters, use the **equals** condition.

   A regular expression is a complex and powerful method for matching text strings. To search using a regular expression, use the **regular expression** condition. A regular expression search is case sensitive. For example, to find all objects that contain the word "Car" search for the string "\Car", without the quotation marks.

   The value for the **Condition** box is saved from session to session.

5. In the **Search in** list, select the part of the model hierarchy that you want to search.

   The value for the **Search in** box is saved from session to session.

6. In the **Class** list, select the single class of objects that you want to search.

7. In the **Property** list, select the type of property that you want to search.

   The (**All Properties**) property searches all properties. The **Object Name** property restricts the search to the name of each object. The **Text Properties** property searches the set of properties that contain text strings, such as Description or Screen Tip, but not including the object name.

   The value for the **Property** box is saved from session to session.

8. Click **Search**.

   The results are listed at the bottom of the **Search** tab. Search results can contain multiple values, such as text properties, for a single object. When you click an object that has multiple values, all the values that belong to that object are selected. If your model contains multiple languages, the search results include properties for each language in the model.

   After you do one search, the **Subset** check box becomes available. If you select the **Subset** check box, the next search operates on the existing search results. The **Subset** check box is cleared after each search. You can do successive subset searches by selecting the **Subset** check box.

9. To see an object in the **Project Viewer**, click an object in the **Search** tab.

10. To see an object in the diagram, right-click an object in the **Search** tab and click **Locate in Diagram**.

You can replace values for properties (p. 40).

The Dependencies Tab

The **Dependencies** tab shows the objects that are dependent on a selected object. For more information, see "Show Object Dependencies" (p. 294).

Naming Conventions for Objects in a Project

All objects in a project must have a unique identifier. The reference can consist of one or more parts, depending upon the type of object. The parts include

- an object name
- a location in the project hierarchy, as expressed in the default language of the project.

Note: If you want two dimensions or query subjects to have the same name in a project, they must be in different namespaces.

One-part Identifiers

Some objects in a project have a one-part identifier. The one-part identifier must be unique across the entire project, even if the namespace contains other namespaces. These objects have a one-part identifier:

- namespaces
- functions
- shortcuts to namespaces
- shortcuts to folders

Two-part Identifiers

Some objects in a project have a two-part identifier consisting of the name of the containing namespace and the name of the object. The object name must be unique in the containing namespace (p. 32). These objects have a two-part identifier:

- regular dimensions
- measure dimensions
- query subjects
- shortcuts to query subjects

For example, a go_sales namespace contains a query subject named Product. The Product query subject has the following name, where the square brackets and periods are the syntax that Framework Manager uses for object identifiers:

```
[go_sales].[Product]
```

Three-part Identifiers

Some objects in a project have a three-part identifier based on the identifier of the containing query subject. Each name must be unique in the containing query subject. These objects have a three-part identifier:

- hierarchies
- measures
- query items

For example, a go_sales namespace contains a query subject named Product, and a query item named Product Code. The Product Code query item has the following name, where the square brackets and periods are the syntax Framework Manager uses for object identifiers:

```
[go_sales].[Product].[Product Code]
```

Four-part Identifiers

Levels in a project have a four-part identifier consisting of the namespace name, the dimension name, the hierarchy name, and the level name.

For example, a go_data_warehouse namespace contains a dimension named Account. A hierarchy in Account is Balance sheet, which contains a level named Account. The Account level has the following name, where the square brackets and periods are the syntax Framework Manager uses for object identifiers:

```
[go_data_warehouse].[Account dimension].[Balance sheet].[Account]
```

Five-part Identifiers

Some objects in a project have a five-part identifier consisting of the namespace name, the dimension name, the hierarchy name, the level name, and the query item name. Five-part identifiers are also used for captions and business keys in member unique names (p. 122).

For example, a go_data_warehouse namespace contains a dimension named Account. A hierarchy in Account is Balance sheet, which contains a level named Account. The Account level contains a query item named Account name. The Account name query item has the following name, where the square brackets and periods are the syntax Framework Manager uses for object identifiers:

```
[go_data_warehouse].[Account dimension].[Balance sheet].[Account].[Account
name]
```

Sample Models

Several sample models are included with Framework Manager for you to explore. The sample models have been created using the guidelines for modeling metadata (p. 321).

In each sample model, the query items have default formatting defined. Names and descriptions were translated into many different languages. By using the Language_lookup parameter map, each user automatically sees folder and item names and descriptions in their preferred language.

In addition to the following sample models, there is the Great Outdoors Company model. This is a data source connection to the Great Outdoors cube.

The Great Outdoors Warehouse Model

This model contains financial information and human resources information for the fictional company, The Great Outdoors. The model accesses a dimensional relational data source.

This sample model is located in *c8_location*\webcontent\samples\models\great_outdoors_warehouse.

The sample model contains these views:

- Database view

  Contains the objects that were imported from the data source. These objects are grouped into namespaces for each business area and one named GO for the tables that are common to all business areas.

  In addition, model query subjects, shortcuts, and filters are organized into namespaces. The namespaces appear as folders to your users. This organization of information into relevant business categories helps users to locate the information that they require.

  All joins and determinants are defined in this view.

  A few model query subjects were added for those that required determinants and for lookup tables.

- Business view

  Contains model query subjects that represent data in terms of business or application needs.

  Calculations, including ones for language, are defined in this view.

  This view improves model portability and prevents the metadata queries that occur when calculations are made directly into the data source query subjects.

- Dimensional view

  Contains regular dimensions, measure dimensions, and scope relationships that were created in Framework Manager.

- Analysis view

  Contains the part of the model that is visible in the studios for the analysis (dimensional) package.

  Shortcuts to the regular and measure dimensions in the Dimensional view are grouped in star schemas and placed in the root of the model in folders, one for each business area. They are clearly marked with "(analysis)" at the end of the name. They are not in a separate namespace because that would add another level in the metadata tree in the studios.

- Query view

  Contains the part of the model that is visible in the studios for the query and reporting package.

  Shortcuts to the model query subjects in the Dimensional view are grouped in star schemas and are placed in folders, one for each business area. They are clearly marked with "(query)" at the end of the name in the root of the model.

In addition, there are packages in the sample model: one for analysis and one for query and reporting. You cannot use the query and reporting package in Analysis Studio.

The Great Outdoors Sales Model

This model contains sales analysis information for the fictional company, The Great Outdoors. The model accesses a transactional system.

This sample model is located in *c8_location*\webcontent\samples\models\great_outdoors_sales.

The sample model contains these views:

- Database view

  Contains the query subjects that were imported from the data source. Because this is a transactional data source, most of the fact tables do not have keys. In some cases, we used multiple-part keys and in other cases, model query subjects were added to calculate the keys and to resolve ambiguous cardinality.

  All joins and determinants are defined in this view.

- Business view

  Contains model query subjects and reference shortcuts that represent the data in terms of business or application needs. Calculations, including ones for language, are defined in this view.

- Dimensional view

  Contains regular dimensions, measure dimensions, and scope relationships. The dimensions are based on the model query subjects in the Business view.

  This is also where the query subjects are renamed, if needed.

- Analysis view

  Contains the part of the model that is visible in the studios for the analysis (dimensional) package.

  Shortcuts to the regular and measure dimensions in the Dimensional view are grouped in star schemas and placed in the root of the model in folders, one for each business area. They are clearly marked with "(analysis)" at the end of the name. They are not in a separate namespace because that would add another level in the metadata tree in the studios.

- Query view

  Contains the part of the model that is visible in the studios for the query and reporting package.

  Shortcuts to the model query subjects in the Dimensional view are grouped in star schemas and are placed in folders, one for each business area. They are clearly marked with "(query)" at the end of the name in the root of the model.

In addition, there are packages in the sample model: one for analysis and one for query and reporting. You cannot use the query and reporting package in Analysis Studio.

Chapter 3: Importing Metadata from Data Sources

Before modeling in Framework Manager, you must import metadata. You can import metadata from a variety of data sources, both relational and dimensional. You can import into a new model or into an existing one. Before importing, it is important that you determine that the data source contains the data and metadata that satisfy your reporting needs.

To import metadata, we recommend that you do the following:

- Define data sources (p. 47)
- Set data source security (p. 47)
- Learn about the types of data source connections (p. 48)
- Learn how to work with data source connections (p. 58)
- Create a data source connection (p. 61)
- Import metadata (p. 65)

Data Sources

Before you can create models and import metadata, you must define data sources. A data source connection supplies the information that Cognos 8 needs to connect to a database.

Each data source can contain one or more physical connections to databases. The data source connection specifies the parameters needed to connect to the database, such as the location of the database and the timeout duration. A connection can include credential information and signons.

You can secure data sources using Cognos 8 authentication. Cognos 8 authentication respects any security that is also defined within the data source. You can create data source signons to isolate the database logon process from the end users. The signon stores the user ID and password required to access the database. You can also deploy data sources.

For more information about data source connections, see the *Administration and Security Guide*.

Data Source Security

You can define security for data sources using Cognos 8 authentication or data source-specific security. Defining Cognos 8 authentication for a data source does not override any database vendor-specific security.

Depending on the data source, one or more of the following types of Cognos 8 authentication are available:

- no authentication

Cognos 8 logs on to the data source without providing any signon credentials.

- Cognos 8 service credentials

  Cognos 8 logs on to the data source using the logon specified for the Cognos 8 service. Users do not require individual database signons. For production environments, however, individual database signons are generally more appropriate.

- external namespace

  Cognos 8 logs on to the data source with the same credentials used to authenticate to the specified external authentication namespace. The namespace specified must be active, users must be logged on to it prior to accessing the data source, and the credentials used for the namespace authentication must be relevant for the data source authentication.

All data sources also support data source signons defined for the Everyone group or for individual users, groups, or roles. If the data source requires a data source signon, but you don't have access to a signon for this data source, you will be prompted to log on each time you access the data source.

Cognos 8 also respects any security defined for the data source. For example, for Cognos cubes, the security may be set at the cube level. For Microsoft Analysis Server data sources, the security may be set using cube roles.

Types of Data Source Connections

Cognos 8 supports many different types of data sources. The data source connection information may vary for each type of data source you use. The sections following provide information specific to individual data sources. If you require additional information about the parameters to connect to your specific data source, see the vendor documentation for the data source you are using.

In a Framework Manager model, catalogs and schemas are properties that are associated with a data source object. If present, these properties qualify any object that is generated in an SQL statement at run time.

When switching the data source against which a model is mapped, qualification levels are often removed. The Relational Database Management System (RDBMS) applies appropriate name space searches as it prepares the SQL statements it receives. Depending on the vendor, the RDBMS may look in up to two places to resolve a reference to an object. An error results if a reference cannot be resolved.

Framework Manager preserves the names of tables and columns as presented by the RDBMS. These names appear in generated Cognos SQL statements. Cognos uses quotes for the names to preserve case and to avoid conflicts with special characters or keywords.

Cognos Cubes

Cognos 8 provides support for accessing secured Cognos cubes as data sources. These cubes include PowerCubes generated by Transformer 7.3 and later, Cognos Finance multi-dimensional cubes,

and Cognos Planning - Contributor cubes. To access a Cognos cube that is secured to a Series 7 namespace, you must first log into the Series 7 namespace.

Transformer 8.3 supports Cognos 8 security, including the Series 7 namespace. You can also transition custom views to the Cognos 8 namespace. When you connect to a Cognos cube that is secured with Cognos 8 security, you must be logged into the namespace the cube was designed with before accessing the cube.

For information about integrating Cognos Finance multi-dimensional cubes in your Cognos 8 environment, see the Cognos Finance *User Guide*. For information about connecting to the Cognos Planning - Contributor unpublished (real-time) data, see the Cognos Planning *Installation Guide*.

Cognos supports the deployment of Cognos 8.3 PowerCubes in production environments that are secured against a single namespace. PowerCubes that are secured against multiple namespaces are not supported in production environments. For this reason, when you deploy cubes for use in production, ensure that they are secured against one namespace. To do this, select the option **Restrict PowerCube authentication to a single namespace** in the Signon properties. For more information, see the Transformer *User Guide*.

For PowerCubes, the connection information in Cognos Connection provides both a Windows and UNIX path to a cube file. If all of your report servers are installed on Windows computers, you need only use the Windows path in Cognos Connection. Similarly, if all of your report servers are installed on UNIX computers, then you need only use the UNIX path in Cognos Connection.

Note: If you are using Framework Manager, the Windows path to the PowerCube is required.

If you have report servers installed on both Windows and UNIX computers and you want the report server running the request to have access to the PowerCube in both environments, you can use both the Windows and UNIX paths in Cognos Connection. To ensure that the same data is returned regardless of the environment in which the report server accesses the cube, the same cube file must be saved in the locations specified by the Windows and UNIX paths in Cognos Connection.

PowerCubes can only be created in the Linux environment if you are using Transformer 8.3 and a Cognos 8 security provider. However, you can deploy secured Series 7 PowerCubes to Linux computers running as report servers in the Cognos 8 environment. The Content Manager computer must be running on a Series 7-compliant platform.

Tips

- If you are testing the migration of your Series 7 PowerCubes to Cognos 8, you can select the authenticate with **All applicable namespaces** method.
- If a PowerCube that you have already created a connection for has to be updated, you can update the connection in Cognos 8 without affecting users who may be currently accessing the original PowerCube.
- When you use Series 7 PowerCubes as data sources, we recommend that you optimize them for Cognos 8. Optimized PowerCubes provide faster data retrieval at runtime. You optimize PowerCubes using a command line utility named pcoptimizer, which is supplied with Cognos 8. For more information about optimizing PowerCubes, see the Troubleshooting section in this guide.

- If a cube contains security views and you are logged on as Anonymous, you will receive an error message. This is because Cognos 8 cannot validate the user "Anonymous" against the Cognos 8 namespace user credentials. To avoid this situation, log on to the Cognos 8 namespace with a user ID and password, and then publish the cube.

If a cube has secured its data using user classes from a Series 7 namespace, you will have to select the namespace to be used for authentication. If the options are disabled or the namespace is not listed, see "Configuring Cognos 8 Components to Use Cognos Series 7" in the Cognos 8 *Installation and Configuration Guide* for how to identify a Series 7 namespace to your Cognos 8 environment.

Tip: If a PowerCube has been rebuilt, you can update the connection information without affecting current users. For information about updating existing data source connections, see the Cognos 8 *Administration and Security Guide*.

Native Metadata

Cognos 8 supports OLAP data sources as well as relational data sources. The term native metadata refers to objects such as models, packages, and queries that are based on an OLAP data source. A namespace that contains native metadata uses this icon to indicate that it is different from namespaces containing other types of metadata.

OLAP data sources are metadata rich data sources. Explicit modeling for these data sources is not enabled in Framework Manager and the package is published directly to the portal. For more information, see "Publish a Package Based on an OLAP Data Source" (p. 263).

Levels are created using the generation names in the labels. If you want to alter the way levels are named, you can do this by changing the dimension build settings in the application that generated the cube. For more information, see the vendor documentation.

Relational data sources require dimensional modeling to enable them to work in Analysis Studio and to work with drill capabilities in the other studios. For more information about dimensional modeling, see "Dimensions" (p. 115).

If you installed Cognos 8 components on UNIX servers, we recommend that you also locate the file-based data source on a UNIX server. You should then use a UNIX path, such as /servername/cubes/Great Outdoors Company.mdc to access the file. For more information, see "Recommendation - Use Network Paths for File-Based Data Sources" (p. 64).

Compound packages contain both OLAP and relational metadata.

Microsoft SQL Server Data Sources

Depending on the types of Microsoft SQL Server data sources you are using, there are considerations you should keep in mind when defining some types of authentication.

Authentication Using Cognos 8 Service Credentials

When connecting to Microsoft SQL Server using OLE DB, you can select **Cognos 8 Service Credentials** as the signon type for the data source connection. This property instructs Cognos 8 to log on to the SQL Server database using the logon specified for the Cognos 8 service. Users do not require individual database signons. However, all users will authenticate to the database with the

same credentials and will have the same view of the data. For production environments, individual database signons are generally more appropriate.

You should not use a Windows local system account for the Cognos 8 server logon with a Microsoft SQL Server OLE DB data source.

Authentication Using an External Namespace

You can configure Cognos 8 to use a Microsoft Active Directory namespace, where users are prompted for credentials as part of the Cognos 8 logon process. You can configure Cognos 8 to use these same credentials automatically when accessing the Microsoft SQL Server data source. The data source connection for Microsoft SQL Server must be configured for **An External Namespace** and that namespace must be the Active Directory namespace.

You can configure Cognos 8 to use a Microsoft Active Directory namespace and to authenticate users for Cognos 8 using Kerberos authentication and delegation. You can configure Cognos 8 to automatically authenticate the user when accessing the Microsoft SQL Server data source. The following configuration is required:

- The Cognos 8 gateway must be installed on an IIS Web server that is configured for Windows Integrated Authentication.
- Content Manager must be installed on a Windows 2003 or Windows XP server.
- Content Manager, the report server (Application Tier Components), IIS Web server, and the data source server (Microsoft SQL Server) must belong to the same Active Directory domain.
- The data source connection for Microsoft SQL Server must be configured for **An External Namespace** and that namespace must be the Active Directory namespace.
- The report servers are trusted for delegation.

For more information about installation options for the gateway and Content Manager, as well as configuring the namespace and delegating trust, see the Cognos 8 *Installation and Configuration Guide*.

Named sets are imported and stored as read-only calculations in the Framework Manager model. The calculation has a flag that identifies it as a named set, and a property that contains the dimension name.

Microsoft Analysis Services Data Sources

When creating a connection to a Microsoft Analysis Services data source, you must specify a language. By default, the language is set to that of your computer's regional setting. You can have only one language selection for a connection. For more information, search the Microsoft Web site (http://www.microsoft.com) or Microsoft Support Web site knowledge base (http://support.microsoft.com) for "locale identifier property".

If you want to publish a Framework Manager model that uses a different locale setting than a previously published model, you must first restart the Cognos 8 service.

Cognos 8 and PowerPlay Enterprise Server support connectivity to Microsoft Analysis Services only from a Windows platform, and both require the Microsoft Pivot Table client libraries. The following configuration is required:

- Cognos 8 Service must run as a domain account, and this domain account must be trusted for delegation and impersonation because it will present the user's credentials to Microsoft Analysis Services.
- The domain account should be added in local security settings in your Windows operating system administration tools.

  Note: You must restart your server after changing this property.
- The domain account should be a member of the OLAP Administrators group on the Microsoft Analysis Services computer.
- The OLE database for OLAP (ODBO) client must be installed. This is included in the Microsoft Analysis Services installation files.

Authentication to an External Namespace

You can configure Cognos 8 to use a Microsoft Active Directory namespace, where users are prompted for credentials as part of the Cognos 8 logon process. You can configure Cognos 8 to use these same credentials automatically when accessing the Microsoft Analysis Services data source. You must configure the data source connection for Microsoft Analysis Server for **An External Namespace** and that namespace must be the Active Directory namespace.

You can configure Cognos 8 to use a Microsoft Active Directory namespace and to authenticate users for Cognos 8 using Kerberos authentication and delegation. You can configure Cognos 8 to automatically authenticate the user when accessing the Microsoft Analysis Services data source. The following configuration is required:

- The Cognos 8 gateway must be installed on an IIS Web server that is configured for Windows Integrated Authentication.
- Content Manager must be installed on a Windows 2003 or Windows XP server.
- Content Manager, the report server (Application Tier Components), IIS Web server, and the data source server (Microsoft Analysis Server) must belong to the same Active Directory domain.
- The data source connection for Microsoft SQL Server must be configured for **An External Namespace** and that namespace must be the Active Directory namespace.
- The report servers are trusted for delegation.

For more information about installation options for the gateway and Content Manager, as well as configuring the namespace and delegating trust, see the Cognos 8 *Installation and Configuration Guide*.

Named sets are imported and stored as read-only calculations in the Framework Manager model. The calculation has a flag that identifies it as a named set, and a property that contains the dimension name.

ODBC Data Sources

If you are connecting to an ODBC data source that has been set up as a user DSN, ensure that the service using the connection is running as the user who created the data source. System DSNs are not restricted to the user who created the data source.

Note: In order to create and test reports with user DSN data sources, you must set up the Cognos8 service with a user account. User DSN data sources are invisible under a system account.

When you create connections to ODBC data sources, select the vendor-specific data source type, such as Microsoft SQL Server (ODBC), from the **Type** menu in the **New Connection** wizard.

Select ODBC to connect to an ODBC data source type that does not appear in the menu.

Database vendor	Database code
IBM Red Brick	RB
Microsoft SQL Server	SS
NCR Teradata	TD
ODBC	OD

IBM DB2 OLAP and Hyperion Essbase Data Sources

IBM DB2 OLAP and Hyperion Essbase data sources are interchangeable and Cognos 8 works with them in exactly the same way. For example, you can connect to an Essbase data source using a DB2 OLAP connection. When you define the data source, you must choose the specific cube for the data source.

Before connecting to a DB2 OLAP or Essbase data source, some configuration is required if the data source uses scenario dimensions, balanced hierarchies, or measures.

Performance Considerations

You must choose either simple or query outlines when making calls in Essbase API.

Base your choice on the performance that you need and factors such as your network environment, the size of the databases, and the types of queries.

Simple Outline

Use simple outline if you primarily run simple queries and use numerous cubes. If you use this option for large cubes, your queries may take a long time to run. Large temporary files may also be created as Essbase builds the necessary grid. After the grid is created, these queries typically run faster and do not increase network traffic. However, more memory is consumed.

This option is the default behavior in the Cognos environment.

Query Outline

Use query outline if you run numerous, complex, or advanced queries that use Cognos 8 features such as nested filtering or sorting.

To enable this option, edit the qfs_config.xml file in the cognos 8 location\configuration directory. Type the following line at the end of the <parameters> subsection of the DB2OlapODP provider section:

```
<parameter name="UseSecondOutline" value="true"/>
```

Configure Scenario Dimensions

If you connect to a DB2 OLAP data source and it contains a scenario dimension, you must manually configure the scenario dimension so that Cognos 8 recognizes it. Otherwise, Cognos 8 will treat the scenario dimension as a regular dimension.

Step

- In the Hyperion Solutions Essbase Administration Services tool, assign a User Defined Attribute (UDA) named COGNOS_SCENARIO_DIMENSION to the scenario dimension.

Specify Balanced Hierarchies

DB2 OLAP Provider does not determine if a hierarchy is balanced or unbalanced. It considers all hierarchies as being unbalanced by default.

In a balanced hierarchy, each path descends to the same depth while the branches in an unbalanced hierarchy descend to different levels.

Steps to Specify Balanced Hierarchies

1. In the Hyperion Solutions Essbase Administration Services tool, create a special User Defined Attribute (UDA) named COGNOS_HIERARCHY_BALANCED in the outline of the DB2 OLAP database. The UDA is created for the root member of the corresponding dimension containing a balanced hierarchy.
2. Set the attribute to 1.

Specify Measure Formats

To improve the readability of the values reported, you can specify an alternative format string for any measure. Define a UDA for the appropriate members in the Account dimension:

`COGNOS_FORMAT=`*`format_string`*

The *format_string* value can be any one of the predefined number formats listed in the table below. You can use a preset numeric format to show values as millions (M) or thousands (K). For example, 1,801,791 can be shown as 1.8M or 1,801.8K.

The predefined format strings are as follows:

Format Option	Sample Value	Example
General	1000000	1000000
0	1000000	1000000
#,##0	1000000	1,000,000
$0	1000000	$1000000
$#,##0	1000000	$1,000,000
0%	1000000	100000000%
%0	1000000	%100000000
0E+000	1000000	1E+006
0K	1000000	1000K
#,##0K	1000000	1,000K
K0	1000000	K1000
K#,##0	1000000	K1,000
$0K	1000000	$1000K
$#,##0K	1000000	$1,000K
0M	1000000000	1000M
#,##0M	1000000000	1,000M
M0	1000000000	M1000
M#,##0	1000000000	M1,000
$0M	1000000000	$1000M
$#,##0M	1000000000	$1,000M

With the exception of the General format string, you can also preset the number of decimal places to show, using the format_string ~n, where n is the number of decimal places. For example, 1,801,791 can be shown as $1,801,791.00 using the format string $#,##0~2. If you do not want any decimal places, end the format string with ~0.

If your client application uses a different locale, you must replace the Currency ($), Thousands (,) and Decimal (.) symbols in the format_string value for the COGNOS_FORMAT UDA with the corresponding locale symbols in effect for the client application.

If you do not specify the number of decimal places to show, or if the format string does not match one of the predefined values (including locale symbols), the General format string is used by default.

You can apply a different format for each measure. The following illustrates some examples of how you can apply different formatting to different measures:

Measure	Applied Format
Measures (Account dimension)	COGNOS_FORMAT=#,##0
Units	COGNOS_FORMAT=#,##K
Costs	COGNOS_FORMAT=$#,###
Profits	COGNOS_FORMAT=0%

Attributes of a Dimension

You can choose how an attribute of a dimension is represented in a model and all associated packages. An attribute of a dimension can be represented as a separate dimension or as a property (query item) of the dimension with which it is associated. You can set this behavior when creating a package using DB2 OLAP or Essbase data sources.

Steps to Specify Modelling Attributes

1. In Cognos Connection, in the upper-right corner, click **Launch, Cognos Administration**.
2. On the **Configuration** tab, click **Data Source Connections**.
3. Click **More** beside the data source with which you want to create a package.
4. Click **Create a Package**.

   The **New Package Wizard** appears.
5. Enter a name for the package and click **Next**.
6. Select a catalogue and click **Next**.
7. Select a cube and click **Next**.
8. Assign a language specific alias table and click **Next**.
9. Choose a language for the alias table and click **Next**.
10. Choose how to present model attributes. Select **Separate the Dimensions** or **Properties of the dimension they are associated with**. The default selection is **Properties of the dimension they are associated with**.

11. Click **Finish**.

Attributes of a Dimension in IBM DB2 OLAP and Hyperion Essbase Data Sources

You can choose how an attribute of a dimension is represented in a model and all associated packages. An attribute of a dimension can be represented as a separate dimension or as a property (query item) of the dimension it is associated with.

Set this behavior in the Metadata Wizard when you import an IBM DB2 OLAP or Hyperion Essbase data source. In the Metadata Wizard, select either **Separate Dimensions** or **Properties of the dimension they are associated with** (default).

Composite Software

Composite software provides heterogeneous query access by retrieving data from different data sources using a single connection. The data sources are managed by Composite software, and the connections to Composite from Cognos 8 are made using an ODBC system DSN.

For more information about using Composite, see the Composite software documentation. For more information about creating the system DSN and using Composite Information Server with Cognos 8, see Cognos 8 and Composite Information Server *Getting Started* and the Cognos 8 and Composite Information Server *Readme* on the Composite Information Server installation CD.

XML Data Sources

If you create an XML data source, you must use XML as the type of connection and specify the location of the XML document in the connection string.

You can specify the connection string for an XML data source as:

- an HTTP URL that identifies the content store required to connect to the XML document.

  An example is HTTP://xmltestserver.cognos.com/XML/country.xml.

  Ensure that you create a Web alias for the directory that contains the XML file and that you enable directory browsing.

- a file path

  A Windows file path example is *servername*\XML\country.xml.

  A UNIX file path example is /*mount name*/XML/country.xml.

- a local file

  An example is C:\XML\country.xml;VALIDATE=ON.

To access a local file, use a file path that uses platform-specific syntax.

To test an XML connection string, you must type the following code at the end of the string:
`;VALIDATE=ON`

The text of this code is not case sensitive.

Parameterized XML Connection Strings

In an HTTP URL connection string for an XML data source, you can use parameters to send additional information. You can embed a prompt definition string in the parameter component.

If the prompt definition is specified in the report, that value is used. Otherwise, the user is prompted to supply a value. Prompting is not supported for other types of connection strings.

An example of a URL component is addressing_scheme://*network_location*/path;parameters?query#fragment_identifier

Encode the parameter component with the definition string in between two sets of question marks. A prompt cannot cross a component boundary.

An example of a parameterized XML string is http://*My_Network_Location*/*My_Path*/myxml.asp?countrysid=??CanadaPrompt??

Parameterized XML connection strings have these restrictions:

- When a URL component is a prompt, it cannot contain other data.
- Prompts embedded in XML connection strings do not work in Framework Manager. You cannot import data from a parameterized XML connection string.
- When you set up a parameterized XML connection string in Cognos Connection, the Test button does not work.
- Validation of the query specification in Report Studio does not work if you are connected to a parameterized XML connection string.

Working With Data Source Connections

You can customize data source connections to meet the needs of users.

Multiple Data Source Connections

If you have access to more than one data source connection in a data source, you are prompted to select a data source connection when you open a Framework Manager project. You can use multiple data source connections in a single data source to facilitate the migration from one environment to another and maintain the integrity of a project.

For example, you can use multiple data source connections to work with metadata from a test data source. Create a new project, using the GoSales data source connection. Create and modify the objects you want in the project, and test to ensure that the project is modeled the way you want. After you close the session, and reopen the Framework Manager project, you can select the production data source connection. When you publish the package to the Cognos 8 server, your users choose which data source connection they want to use in their report.

Multiple connections to the same data source must be defined in Cognos Connection. If you want to support multiple connections for each data source, clear the data source catalog and schema names, and create a connection for each database in Cognos Connection.

Note: If you are working with multiple cubes containing unlike metadata, we recommend that you use separate data sources for each cube. To be able to expand an OLAP package in the Studios, the internal name of both cubes must be the same. If you want to run saved reports that use different data source connections, the cube name, as well as the dimension, hierarchy, level and attribute names, must be the same in both cubes. If you use a single data source with a separate connection for each cube, the internal names of all the cubes must be the same.

For more information about data source connections, see the *Administration and Security Guide*.

Databases and Sorting Functions

If you use sorting functions such as running-totals in reports, you may need to add a collation sequence to the database connection string to maintain data integrity. Otherwise, when sorting a report that is grouped by one or more columns containing foreign characters, you may get inconsistent results. This is because without the collation sequence information, the default locale is used to sort data, whereas databases may use different collations in their sorting.

To prevent sorting inconsistencies, you must add the correct collation sequence to your database connection string when you create a data source or add a connection.

For example, in Oracle, if you specify the collation sequence as Binary at the database level, you should provide the same collation sequence value in the connection string.

Isolation Levels

The isolation level specifies how transactions that modify the database are handled. By default, the default object gateway is used. Not all types of databases support each isolation level. Some database vendors use different names for the isolation levels.

For OLAP data sources, including SAP BW, the isolation level is read-only.

The following isolation levels are in increasing order of isolation:

- Read Uncommitted

  Changes made by other transactions are immediately available to a transaction.

Database type	Equivalent isolation level
Oracle	Not applicable
DB2	Uncommitted read
MS SQL Server	Read uncommitted
Sybase Adaptive Server Enterprise	Read uncommitted
Informix	Dirty read

- Read Committed

  A transaction can access only rows committed by other transactions.

Database type	Equivalent isolation level
Oracle	Read committed
DB2	Read stability
MS SQL Server	Read committed
Sybase Adaptive Server Enterprise	Read committed
Informix	Committed read

- Cursor Stability

  Other transactions cannot update the row in which a transaction is positioned.

Database type	Equivalent isolation level
Oracle	Not applicable
DB2	Cursor stability
MS SQL Server	Not applicable
Sybase Adaptive Server Enterprise	Not applicable
Informix	Cursor stability

- Reproducible Read

  Rows selected or updated by a transaction cannot be changed by another transaction until the transaction is complete.

Database type	Equivalent isolation level
Oracle	Not applicable Execute the statement, "SET TRANSACTION READ ONLY," prior to starting a transaction
DB2	Repeatable read
MS SQL Server	Repeatable read
Sybase Adaptive Server Enterprise	Repeatable read

Database type	Equivalent isolation level
Informix	Repeatable read

- Phantom Protection

  A transaction cannot access rows inserted or deleted since the start of the transaction.

Database type	Equivalent isolation level
Oracle	Not applicable
DB2	Not applicable
MS SQL Server	Not applicable
Sybase Adaptive Server Enterprise	Not applicable
Informix	Not applicable

- Serializable

  A set of transactions executed concurrently produces the same result as if they were performed sequentially.

Database Type	Equivalent isolation level
Oracle	Serializable
DB2	Not applicable
MS SQL Server	Serializable
Sybase Adaptive Server Enterprise	Serializable
Informix	Not applicable

Create a Data Source Connection

When you create a data source, you must provide the information required to connect to the database. This information is provided in the form of a connection string.

You can include authentication information for the database in the data source connection by creating a signon. Users need not enter database authentication information each time the connection

is used because the authentication information is encrypted and stored on the server. The signon produced when you create a data source is available to the Everyone group. Later, you can modify who can use the signon or create more signons.

New Data Sources

You can create data sources in the portal or in Framework Manager. Because they are stored on the server, data sources appear in both places, regardless of where they were created. Existing data source connections can be edited only in the portal.

If you are an administrator, you can set up all required data sources before models are created in Framework Manager so that all connections are available in the Framework Manager Import wizard.

Data sources are stored in the Cognos namespace and must have unique names. For example, you cannot use the same name for a data source and a group.

Physical Connections

A data source defines the physical connection to a database. A data source connection specifies the parameters needed to connect to a database, such as the location of the database and the timeout duration.

Note: The schema name in the connection string for an Oracle database is case-sensitive. If the schema name is typed incorrectly, you cannot run queries.

Required permissions

Before creating data sources, you must have write permissions to the folder where you want to save the data source and to the Cognos namespace. You must also have execute permissions for the **Data Source Connections** secured feature.

Connections to Specific Data Sources

If you are creating a connection to one of these data sources, review the pertinent information before continuing:

- Cubes from Cognos products (p. 48)
- Microsoft SQL Server (p. 50)
- Microsoft Analysis Services (p. 51)
- ODBC data sources (p. 53)
- Composite data sources (p. 57)

Depending on the type of database you are connecting to, you may also need to review information about isolation levels (p. 59) and sorting functions (p. 59).

For information about data source security, see the Cognos 8 *Administration and Security Guide*.

Steps

1. If you are not already using the Run Metadata Import wizard, click the namespace, folder, or segment you want to import into, and from the **Actions** menu, click **Run Metadata Wizard**.
2. In the **Select Metadata Source** window, click **Data Sources**.
3. Click **Next**.
4. In the **Select Data Source** window, click **New**.
5. In the name and description page, type a unique name for the connection and, if you want, a description and screen tip, and then click **Next**.
6. In the connection page, click the type of database to which you want to connect, select an isolation level, and then click **Next**.

   The connection string page for the selected database appears.
7. Enter any parameters that make up the connection string, and specify any other settings, such as a signon or a timeout.

   One of the following options may apply depending on the data source to which you are connecting:

   - If you are connecting to a Cognos cube, you must enter the full path and file name for the cube. An example for a local cube is C:\cubes\Great Outdoors Company.mdc. An example for a cube on your network is *servername*\cubes\Great Outdoors Company.mdc. If you leave the **Read cache size (MB)** field blank or enter 0, Cognos Connection uses the default value in the ppds_cfg.xml file in the configuration folder.

     You must also select an authentication method. Click **All applicable namespaces**, or click to authenticate using an unsecured PowerCube or to a PowerCube secured against many namespaces. Click **Restrict PowerCube authentication to a single namespace**, and select a namespace from the list. If you are connecting to a password protected PowerCube, click **Cube Password**, and then type the password in the **Password** and **Confirm Password** boxes.

     Note: Select **All applicable namespaces** only if you are migrating Series 7 PowerCubes to Cognos 8 in your development or test environment. This setting can also be used for unsecured PowerCubes that are in a production environment. For more information, see the Transformer *User Guide*.

   - If you are connecting to an ODBC data source, the connection string is generated from the name you enter in the **ODBC data source** box and any signon information. The data source name is an ODBC DSN that has already been set up. You can include additional connection string parameters in the **ODBC connect string** box. These parameters are appended to the generated connection string.

   - If you are connecting to a Microsoft Analysis Services data source, select an option in the **Language** box. If you selected **Microsoft Analysis Services 2005**, you must specify an

instance name in the **Named instance** since you can have more than one instance on each server.

- If you use a Microsoft Active Directory namespace and you want to support single signon with Microsoft SQL Server or Microsoft Analysis Server, select **An External Namespace**, and select the Active Directory namespace. For more information about configuring an Active Directory namespace, see the *Installation and Configuration Guide*.
- If you selected **Other Type** as the data source type, you will have to build the connection string manually.

Tip: To test whether parameters are correct, click **Test**. If prompted, type a user ID and password or select a signon, and then click **OK**. If you are testing an ODBC connection to a User DSN, you must be logged on as the creator of the DSN for the test to succeed.

8. Click **Finish**.
   - If you created an OLAP data source, a confirmation appears and you are given the option to create a package using your new data source. Click **OK** to return to **Data Source Connections**, or click **Create a Package** and **OK**, and follow the instructions in the new package wizard.

     Note: You can create a package with your new data source at a later date if you do not want to do it now.
   - If you selected any other type of data source, the new data source appears in **Data Source Connections** on the **Configuration** tab, and can be selected when using the Import Wizard in Framework Manager.

Tip: To test a data source connection, right-click the data source in the **Data Sources** folder and click **Test**.

If you created a signon, you can now modify the properties of the signon or add more signons.

Recommendation - Use Network Paths for File-Based Data Sources

If you have a distributed installation with several servers, we recommend that you use network paths for all file based data sources rather than local paths. This will ensure that the data sources can be accessed by the services that require them, regardless of which server requires the data.

When you create a connection to a file-based data source, such as a PowerCube, you enter a path and filename. You can use a local path, such as C:\cubes\Great Outdoors Company.mdc, or a network path, such as *servername*\cubes\Great Outdoors Company.mdc, to point to the file.

In a distributed installation, where report servers are running on different computers, using a local path would require that the file and path be valid on each computer where a report server is running. Alternatively, if you use a network path to point to a file, each report server would point to the same file on the network without having the file available locally. Also, to ensure that the file is always available, we recommend that you store it in a shared directory that can be accessed on your network.

If you have installed Cognos 8 components on UNIX servers, we recommend that you also locate the file-based data source on a UNIX server. You should then use a UNIX path, such as */servername/* cubes/Great Outdoors Company.mdc to access the file.

If you have installed all components on a single computer, you can use local paths, but you must ensure that the services requesting the data have the appropriate access to the data files on the computer.

For Windows distributed installations, we recommend that you use UNC paths to shared directories for any file based data source, such as PowerCubes or XML files.

Importing Metadata

You can import metadata into a new project or an existing project. Importing metadata is an operation that can be performed many times to extend the project. You can also export your model to a Common Warehouse Metamodel (CWM) "Export Metadata" (p. 297).

Framework Manager can use the metadata and data from external data sources to build a project.

To import metadata, you must indicate which sources you want and where they are located. You can import from only one data source at a time. If you want to import from more than one data source, you must perform multiple imports.

If you want to add a project to a repository, you must set up the repository connection "Creating Repository Connections" (p. 280).

You can import metadata from

- relational databases, such as Oracle, DB2, and Microsoft SQL Server
- SAP BW data sources
- Cognos 8 models
- Architect models and Impromptu catalogs
- DecisionStream or Cognos 8 Data Manager models
- third party metadata sources
- XML as a data source

For information about the data source types supported by Cognos, see the Cognos support Web site (http://support.cognos.com).

For information about working with data source connections, see (p. 58).

Duplicate Object Names

When you import metadata, you can select how you want the import to handle duplicate object names. You have the option of not importing the object, or importing and creating a unique name. The advantage of importing everything except these duplicate objects is that you can add new database objects to the project without specifying them individually, and without going through

synchronization "Synchronize Projects" (p. 306). To import metadata that has the same table names, you must create two namespaces and import each table into a different namespace.

When you import SAP BW metadata, Framework Manager assigns a unique name to each object. Therefore, if you rename an object in the model and then reimport it, Framework Manager recognizes that it already exists. To reimport an object with a different unique name, you can create a new namespace and reimport the object into this namespace.

You can also import objects that have the same name (p. 78).

Import Metadata from a Relational Database

When you import metadata from a relational database, you can import all the metadata or select particular object types such as tables, columns, views, synonyms, stored procedures, and functions. You can also import system objects from a relational source. System stored procedures are not supported. Framework Manager supports only user-defined stored procedures.

The following table shows the database objects that are mapped to Framework Manager objects.

Database object	Framework Manager object
table	query subject
column	query item
view	query subject
synonym	query subject
procedure	query subject
function	project function

Steps

1. Click the namespace, folder, or segment that you want to import into and, from the **Actions** menu, click **Run Metadata Wizard**.
2. Follow the instructions in the **Metadata Wizard**:
   - Select a data source connection and click **Next**.

     If the data source connection you want is not listed, you must first create it (p. 61).
   - Select the check boxes for the objects you want to import.
   - Specify how the import should handle duplicate object names. Choose either to import and create a unique name, or not to import. If you choose to create a unique name, the imported object appears with a number. For example, you see QuerySubject and QuerySubject1 in your project.

- If you want to import system objects, select the **Show System Objects** check box, and then select the system objects that you want to import.
- Specify the criteria to use to create relationships.
- If you want to convert all cardinalities to `1`, clear the **Fact detection enabled** check box. Removing `n` cardinalities disables fact detection and the ability to automatically prevent double-counting. For more information, see "Modeling 1-n Relationships as 1-1 Relationships" (p. 355).

3. Click **Import**.

   Import statistics including a list of objects that could not be imported and a count of objects that were imported are shown.

4. Click **Finish**.

After importing metadata, you must check the imported metadata for the following areas:

- relationships and cardinality (p. 338)
- determinants (p. 94)
- the **Usage** property for query items (p. 148)
- the **Regular Aggregate** property for query items (p. 148)

For more information about checking the metadata, see "Verifying Imported Metadata" (p. 338).

Import Metadata from a Cognos 8 Model

You can import metadata from an existing Cognos 8 model.

Note: If you import from another Framework Manager project, expression syntax is not adjusted for each language. For example, you create a Framework Manager project using French as the design language and you use French-specific syntax in calculations and filters. You then create a new project using English as the design language and you import the French project into the new project. Expressions defined in the calculations and filters are not valid. You must manually modify the expression after importing the metadata.

Steps

1. Click the namespace, folder, or segment you want to import into and, from the **Actions** menu, click **Run Metadata Wizard**.
2. Click **Cognos 8 Model** and click **Next**.
3. Locate the Cognos 8 model (.cpf file) that you want, click **Open,** and click **Next**.
4. Follow the instructions in the **Import** wizard:
   - Select the check boxes for the objects that you want to import.
   - Specify how the import should handle duplicate object names.

Choose either to import and create a unique name, or not to import. If you choose to create a unique name, the imported object appears with a number. For example, you see QuerySubject and QuerySubject1 in your project.

5. Click **Next** and click **Finish**.

Import Metadata from an Architect Model or an Impromptu Catalog

To import metadata from an Architect model or an Impromptu catalog, you must first convert it to XML files. Because of differences between Cognos Series 7 and Cognos 8, after you import the migrated metadata in Framework Manager, additional work is required to test and refine the metadata.

For information about the migration of specific Series 7 objects, see the Cognos 8 Migration and Upgrade *User Guide* on the Cognos support site (http://support.cognos.com).

Steps

1. Ensure that you exported the Architect model or Impromptu catalog.
2. Click the namespace, folder, or segment you want to import into and, from the **Actions** menu, click **Run Metadata Wizard**.
3. Click either **Cognos Architect (.xml)** or **Cognos Impromptu (.xml)** and click **Next**.
4. Locate the Architect or Impromptu XML file that contains the metadata to import.

   A message in the **XML Preview** window confirms that you chose a valid XML file.
5. Click **Open**.
6. Select the namespace containing your Series 7 security information.
7. Click **Import**.

   A list of created objects appears.
8. If you want to verify the imported metadata, click the **Verify after import** check box.
9. Click **Finish**.

Import Metadata from DecisionStream or Cognos 8 Data Manager

You can use Framework Manager to import metadata from an XML file created by DecisionStream or Cognos 8 Data Manager. You can import

- a physical layer residing in the Physical Metadata namespace

  This layer contains data source query subjects representing the imported tables. The physical layer contains query subjects and physical relationships between query subjects. These physical relationships are inferred from the relationships defined in the import file. Imported tables become Framework Manager query subjects, and surrogate keys become Framework Manager determinants.

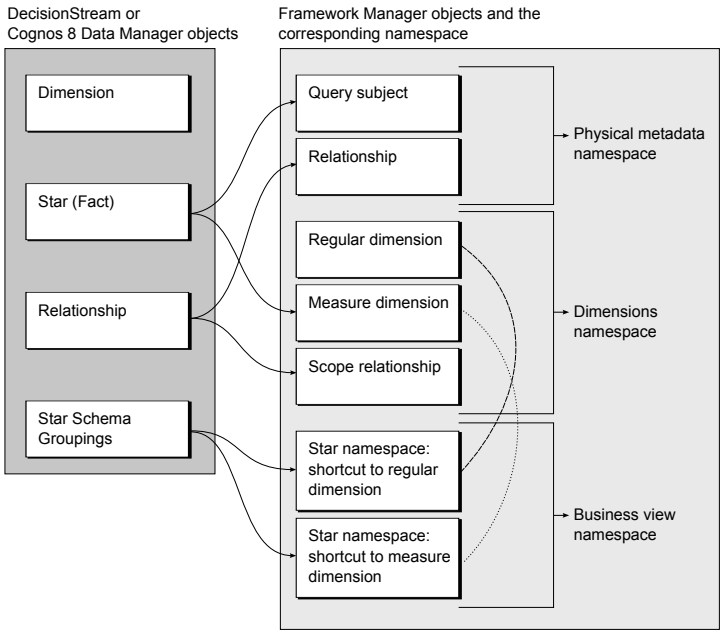
- a dimensional layer residing in the Dimensions namespace

  This layer contains regular dimensions, measure dimensions, and scope relationships. The regular dimensions may be conformed or non-conformed. The measure dimension objects correspond to the imported stars. The scope relationships are inferred from the relationships defined in the import file.

- a logical layer residing in the Business View namespace

  This layer contains shortcuts to the regular and measure dimensions in the Dimension namespace. The shortcuts are organized as star schema groupings, which are namespaces with the same name as the stars from the import file.

The following diagram shows how objects from DecisionStream and Cognos 8 Data Manager are mapped to Framework Manager objects.



Facts

A star maps to a Framework Manager query subject in the Physical Metadata namespace or as a measure dimension in the Dimensions namespace. The following fact attributes are included in the model.

Attribute name	Framework Manager mapping
Table name	Name of the database query subject on which the model query representing the fact is based

Attribute name	Framework Manager mapping
Short name	Query subject description
Business name	Custom property
Description	Query subject description
Column name	Query item name
Column type	Query item data type
Column length	Query item size
Column short name	Custom property
Column business name	Custom property
Column description	Query item description
Column type	Query item usage
Table keys	Determinants in the Physical Metadata namespace

Connections

A connection maps to a Framework Manager data source.

Note: A data source connection is not automatically created in Content Manager. You must manually create the connection in Content Manager as explained in the import procedure steps (p. 68).

The following data source attributes are included in the model.

Attribute name	Framework Manager mapping
Connection short name	Custom property
Connection business name	Data source name
Connection description	Data source description
Connectivity	type.interface property
Connection string	Custom property

Dimension Builds

A dimension build maps to Framework Manager as a top-level namespace.

Hierarchies

A dimension containing hierarchies, levels, and columns maps to a Framework Manager regular dimension containing hierarchies, levels, and query items.

Conformed Stars

Conformed stars map to a Framework Manager namespace that resides in the Business View namespace. It contains shortcuts referencing the dimensions.

The following conformed star attributes are included in the model.

Attribute name	Framework Manager mapping
Star short name	The name of the namespace representing the star
Star business name	The name of the measure dimension representing the fact
Star description	The description of the measure dimension representing the fact
Facts	Shortcuts to a measure dimension
Dimensions	Shortcuts to regular dimensions
Hierarchies	Hierarchies in the regular dimension representing the DecisionStream dimensions

Model Properties

The export file contains the following model properties.

Attribute name	Framework Manager mapping
Schema version	Not mapped
Catalog version	Custom property
Model short name	The name of the namespace representing the top-level model object
Model business name	Custom property

Attribute name	Framework Manager mapping
Model description	The description of the namespace representing the top-level model object

Steps

1. Click the namespace, folder, or segment that you want to import into and, from the **Actions** menu, click **Run Metadata Wizard**.
2. Click **Cognos DecisionStream (.xml)** or **Cognos 8 Data Manager (.xml)** and click **Next**.
3. Locate the XML file that contains the metadata to import.

   A message in the **XML Preview** window confirms that you chose a valid XML file.
4. Click **Open** and then click **Import**.

   A list of created objects appears.
5. If you want to verify the imported metadata, click the **Verify after import** check box.
6. Click **Finish**.
7. Create the data source connection in Content Manager. The name you define in Content Manager must be the same as the data source name shown in the **Data Sources** folder in Framework Manager.

   For information about how to create a data source connection, see (p. 61).

Import Metadata From a Third Party Metadata Source

You can use Framework Manager to import metadata from a third party source. Metadata is imported using a metadata bridge.

You can import both third party metadata and relational metadata into the same model. We recommend that you start with a new Framework Manager model and import the third party metadata before the relational metadata. This avoids conflicts if you import objects that have the same name.

During a third party import, data sources are created based on information provided through the import wizard. If at least one physical object in the third party source references a database schema or catalog or both, one Framework Manager data source is created with its catalog or schema properties set to the names of the catalog or schema defined in the metadata. A generic data source is created for those physical objects that do not reference a catalog or schema.

If you want to access metadata from a third party data source, you must perform a physical model import.

Not all third party data sources contain metadata that is appropriate for business intelligence reporting and not all third party concepts map to Framework Manager. The metadata import is tailored to Framework Manager and only compatible metadata will be imported.

All metadata bridges deliver a physical layer that provides the basis for further modeling. The richness of the resulting Framework Manager model is directly related to the richness of the metadata source.

Multiple Databases

Third party metadata sources can be based on multiple databases. The best way to import these multiple data sources into Framework Manager is to perform multiple imports. For each import, you select the items that correspond to that specific data source.

For example, the first time that you import from a third party metadata source, you select datasource1 and all the items that correspond to that data source. The next time, you select datasource2 and the items that correspond to that data source. You continue to import until you have imported all the data sources and their corresponding items.

Third Party Options

Metadata is extracted from the third party data source by the Meta Integration® Model Bridge (MIMB). Not all options apply to Framework Manager. For information about supported tools and object mappings, see the Meta Integration Web site.

Framework Manager Options

The Framework Manager options available are the same regardless of the type of metadata source that you select. This table shows the options used to create objects in Framework Manager.

Framework Manager options	Description
Logical / Physical Representation	Specifies how logical and physical objects are represented. **Integrated** represents the logical and physical objects as one integrated object. This is the default. **Separated** represents the logical and physical objects as two related objects. **Separated (verbose)** represents the logical and physical objects as two distinct objects.
Diagram Representation	Specifies whether each diagram is represented as a namespace, a package, or both. Both is the default. **None** indicates that diagrams are not represented in the project.
Namespace Hierarchy	Specifies whether the hierarchy of packages is kept in the logical namespace, in the physical namespace, or both. Both is the default. **None** indicates that the hierarchy of packages is not retained.

Framework Manager options	Description
Export Logical Only Classes	Indicates whether logical only classes are extracted along with all their attributes. The default value is Yes.
Export Logical Only Attributes	Specifies whether logical only attributes are extracted. **Drop** indicates that logical information of a class is not extracted if it contains a logical only attribute. **Export** indicates that logical only attributes are extracted. This is the default. **No** indicates that logical only attributes are not extracted.
Encoding	Specifies the character set of the model. **ISO-8859-1** is the Latin 1 encoding. This is the default on all platforms except Windows. **Ks_c_5601-1987** is the default Korean encoding. **Shift_jis** is the default Japanese encoding. **utf_8** is the UTF8 encoding. **Windows-1252** is the default Windows encoding. This is the default on Windows platforms.
Consistency Check	Specifies the consistency check level. **Basic** is the recommended consistency check level. **Extensive** performs a more thorough validation of the model. **None** indicates that no validation is performed.

Before you can import metadata, there must be a connection to the data source (p. 61).

Steps to Import from a Third Party Metadata Source

1. Click the namespace, folder, or segment you want to import into, and from the **Actions** menu, click **Run Metadata Wizard**.
2. Select **Third Party Metadata Sources** and click **Next**.
3. Click the metadata type to import.
4. In the **File Name** box, browse to locate the file that contains the metadata to import.

   A message in the **XML Preview** window confirms that you chose a valid XML file.
5. Click **Open** and then click **Next**.
6. In the **Third Party Specific Import Options** dialog box, click the options that you want.

The options that you see are based on the selected third party data source.

Note: We recommend that you use the default options. These default options optimize the third party metadata import. If you change the options, you may see unexpected results. To revert back to the default options, click **Use Defaults**.

7. Click **Next**.
8. In the **Framework Manager Specific Import Options** dialog box, click the options that you want, and click **Next**.
9. Follow the instructions in the **Metadata Wizard**:
   - Select a data source connection and click **Next**.

     If the data source connection you want is not listed, you must first create it (p. 61).
   - Select the check boxes for the objects you want to import.
   - Specify how the import should handle duplicate object names. Choose either to import and create a unique name, or not to import. If you choose to create a unique name, the imported object appears with a number. For example, you see QuerySubject and QuerySubject1 in your project.
   - Specify the criteria to use to create relationships and click **Import**.

     For more information, see "Relationships" (p. 79).

   Import statistics including a list of objects that could not be imported and a count of objects that were imported are shown.
10. Click **Finish**.

Modeling After a Third Party Import

The source metadata has a logical structure that is compatible with Cognos 8. During the import, this structure is preserved for higher fidelity with the source model.

After the import, only the physical metadata is available.

Do the following:

- ❑ analyze and review the model, see "Analyze a Model" (p. 185)
- ❑ set the determinants
- ❑ set the **Usage** property

  Some bridges require facts to be set manually because this is not available from the metadata source.
- ❑ verify the relationships and cardinality
- ❑ resolve ambiguous relationships, such as multiple valid relationships, reflexive relationships, and recursive relationships

Troubleshooting Metadata from Third Party Sources

This section describes issues that you can encounter when working with metadata that you have imported into Framework Manager.

Cannot Test a Query Subject from a Third Party Source

You cannot test a query subject that you have imported from a third party data source.

For example, you test a query subject and get this error message:

QE-DEF-0177 An error occurred while performing operation 'sqlPrepareWithOptions' status='-201'.

UDA-SQL-0196 The table or view "GOSALES1.COGNOS.COM.GOSALES.CONVERSION_RATE" was not found in the dictionary.

To resolve this problem, ensure that none of the following conditions exist:

- The Framework Manager data source object created by the third party import, and referenced by the query subject, does not have an identically named corresponding data source in the Cognos 8 Business Intelligence content store.
- The corresponding content store data source exists but is not valid (the connection information is wrong).
- The optional schema or catalog properties of the data source object that the third party import created and the query subject references are incorrect (for the databases accepting case-sensitive identifiers, ensure that the case is also correct).
- The database object (table or view) represented by the query subject no longer exists in the database or it was renamed after the third party model was created, or it is otherwise not accessible within the current connection.
- The **Query Type** is invalid. This can happen if you import a query subject from SAP BW and change the value of the **Query Type** property for the data source. By default the value of **Query Type** is **multidimensional**. Do not change it.

Relationships Involving Table Views Are Not Imported from an Oracle Designer File

Primary key and foreign key relationships involving at least one table view are not imported from an Oracle Designer file into Framework Manager.

Create the primary key and foreign key relationships manually.

SQL View Definitions Are Not Imported from a Power Designer File

The views in Power Designer must be linked to the table that they reference. Otherwise the information is not saved in the Power Designer file and cannot be imported.

To resolve this problem, see the tooltip for the third party import bridge for Power Designer.

Some Expressions Imported from Third Party Metadata Sources Are Not Valid

Support for expression parsing was improved in the MIMB in Cognos 8. Functions that are equivalent between third party metadata sources and Cognos are parsed. Exceptions to this include

- functions with no mapping between Cognos and the third party metadata source

  For example, a Business Objects expression that is problematic is `@select Country) LIKE 'C%'`.

- expressions that use a specialized syntax

  For example, DB2 Cube Views use DB2CV references ($$1, $$2, and so on).

Solution

If you have functions with no mapping between Cognos and the third party metadata source, after importing expressions from third party metadata sources, edit these expressions manually in Framework Manager to conform to the equivalent Framework Manager syntax.

If you have expressions that use a specialized syntax, do the following:

1. In Framework Manager, identify all imported query items that represent embedded calculations using references such as $$1, $$2, and so on.
2. In the **Properties** pane, find the information provided in the **patternMapping** property for each query item.

   The **patternMapping** property indicates the mapping between the parameters in the imported calculation and the actual object references in the original model.
3. Double-click a broken query subject.
4. Double-click the embedded calculation corresponding to the query item identified in step 1.
5. If the calculation was assigned a default name ("Calculation..."), replace it with the actual query item name.
6. Replace the imported parameters with the actual Framework Manager object references that these parameters represent.
7. Repeat these steps for each broken query subject.

Import Metadata Using XML as a Data Source

You can import an XML file as a tabular data source in Framework Manager. You can import it locally or from a remote site through a valid URL. In Framework Manager the XML file is used to model metadata and create a package.

The XML file is validated and parsed at run time, when the query is processed by either Report Studio or Query Studio. If you add the VALIDATE=ON option to the connection string, Framework Manager partially validates the XML file in the <columnList> tag that describes the metadata. For information about supported data types, see "XML Data Types" (p. 569).

You must use the xmldata.xsd schema to validate the XML file. The schema is located in the \c8\ bin folder. It is not necessary to specify the location of the schema in the XML file itself.

To use XML as a data source, ensure that

- you do not use Native SQL to access data in an XML file
- you do not access Binary Large Objects (BLOB)
- you use only `sqlColumns()` and `sqlTables()` metadata calls

  Other calls return an unsupported function error.
- the XML file is well-formed and valid

Before you import XML as a data source, there must be a connection to the data source (p. 61). If the XML data source is on another computer, you must use an account that has permissions to access the data source.

To use XML as a data source, you must understand XML, schemas, and other XML-related technology.

Before you can import metadata, there must be a connection to the data source (p. 61).

Steps

1. Click the namespace, folder, or segment that you want to import into, and from the **Actions** menu, click **Run Metadata Wizard**.
2. Click the XML data source that you want to import, and click **Next**.

Import Objects with the Same Name

Namespaces are containers like folders. Objects in a Framework Manager project must be uniquely identifiable. If you have two objects that have the same name, they must reside in two separate namespaces.

For example, you have a database that contains financial data. One set of tables represents Forecast and Actual information. Both the Forecast and Actual information have tables named Accounts Payable and Accounts Receivable. To import these tables into Framework Manager and use the same table names in the project, you must create two namespaces. You can name one namespace Forecast, and the other namespace Actual.

Steps

1. Click the model or root namespace and from the **Actions** menu and click **Create**, **Namespace**.
2. Right-click the namespace, click **Rename,** and type a descriptive name.

Chapter 4: Modeling Relational Metadata

Note: Information on SAP BW metadata is in "Working with SAP BW Metadata" (p. 191).

After importing metadata, you must ensure that it is set up to meet your users' reporting requirements, and provide any additional information that they require. Enhancements you make in Framework Manager do not affect the original data source.

Tip: To verify that the model meets the reporting requirements, you can select objects that will appear in a report and test them. The test results show you the report that your users will see as well as the SQL and messages from Cognos 8, if any. Or you can publish a package at any time and then use the package to create reports.

You can check the project at any time to ensure that the references between the objects it contains are valid (p. 247).

We recommend that you create at least two views: the import view and the business view. Using two, or more, views makes it easier to remap items to a new data source (p. 295).

The import view contains the metadata you imported from the data source. To ensure that the metadata is set up correctly in the import view, do the following:

- Ensure that the relationships reflect the reporting requirements (p. 79).
- Optimize and customize the data retrieved by query subjects (p. 87).
- Optimize and customize the data retrieved by dimensions (p. 115). You may want to store dimensions in a separate dimensional view.
- Handle support for multilingual metadata (p. 131).
- Control how data is used and formatted by checking query item properties (p. 139).

The business view provides a layer to the information in the source data so that it is easier for your users to build reports. To enhance the metadata in the business view, do the following:

- Add business rules, such as calculations and filters, that define the information users can retrieve (p. 153).
- Organize the model by creating separate views for each user group that reflect the business concepts familiar to your users (p. 175).

You can analyze the metadata to ensure that the model is following current modeling guidelines by using the **Model Advisor**(p. 185).

Relationships

A relationship describes how to create a relational query for multiple objects in the model. Without relationships, these objects are isolated sets of data.

Relationships work in both directions. You often must examine both directions to fully understand the relationship.

The different types of relationships are

- one-to-one

  One-to-one relationships occur when one instance of data in a query subject relates to exactly one instance of another. For example, each student has one student number.

- one-to-many or zero-to-many

  One-to-many or zero-to-many relationships occur when one instance of data in a query subject relates to many instances of another. For example, each teacher has many students.

- many-to-many

  Many-to-many relationships occur when many instances of data in a query subject relate to many instances of another. For example, many students have many teachers.

When importing metadata, Framework Manager can create relationships between objects in the model based on the primary and foreign keys in the data source. You can create or remove relationships in the model so that the model better represents the logical structure of your business.

After you import metadata, verify that the relationships you require exist in the project and that the cardinality is set correctly. The data source may have been designed without using referential integrity. Often, many primary and unique key constraints are not specified. Without these constraints, Framework Manager cannot generate the necessary relationships between fact tables and dimension tables.

Framework Manager stores relationships in the nearest common parent of the objects that participate in the relationship. The parent can be either a folder or a namespace. If you move one of the participating objects outside the common parent, the relationship moves to the next namespace that is common to both ends of the relationship. If you move a relationship to a different folder or namespace, the participating objects also move to the same folder or namespace.

Tip: Use the **Search** tab (**Tools** pane) to find an object of class Relationship whose name matches a specified pattern. For example, if you search for a relationship whose name contains Order Header, Framework Manager finds all relationships that have Order Header as one end. If you renamed a relationship, a search of this type may not find it.

Cardinality

Relationships exist between two query subjects. The cardinality of a relationship is the number of related rows for each of the two query subjects. The rows are related by the expression of the relationship; this expression usually refers to the primary and foreign keys of the underlying table.

Cognos 8 uses the cardinality of a relationship in the following ways:

- to avoid double-counting fact data
- to support loop joins that are common in star schema models
- to optimize access to the underlying data source system

- to identify query subjects that behave as facts or dimensions

A query that uses multiple facts from different underlying tables is split into separate queries for each underlying fact table. Each single fact query refers to its respective fact table as well as to the dimensional tables related to that fact table. Another query is used to merge these individual queries into one result set. This latter operation is generally referred to as a stitched query. You know that you have a stitched query when you see `coalesce` and a full outer join.

A stitched query also allows Cognos 8 to properly relate data at different levels of granularity (p. 327).

You must ensure that all relationships and cardinality correctly reflect your users' reporting requirements.

For more information, see "Cardinality in Generated Queries " (p. 322) and "Cardinality in the Context of a Query" (p. 323).

Detecting Cardinality from the Data Source

When importing from a relational data source, cardinality is detected based on a set of rules that you specify. The available options are

- use primary and foreign keys
- use matching query item names that represent uniquely indexed columns
- use matching query item names

The most common situation is to use primary and foreign keys as well as matching query items that represent uniquely indexed columns. The information is used to set some properties of query items as well as to generate relationships.

To view the index and key information that was imported, right-click a query subject and click **Edit Definition**. For a query subject, you can change the information in the **Determinants** tab.

Optional relationships, full outer joins, and many-to-many relationships can be imported from your data source. Framework Manager will run them as queries.

Note: All regular dimensions begin as query subjects. If you converted a query subject to a regular dimension, note that determinant information for the query subject is leveraged as a starting point to define the levels of a single hierarchy. We recommend that you review the levels and keys created in the hierarchy of the dimension.

Notation

By default, Framework Manager uses Merise notation. Merise notation marks each end of the relationship with the minimum and maximum cardinality of that end. You can also use Crowsfeet notation, which provides a pictorial representation of the relationship. For information about how to change the notation, see "Change the Settings for Diagrams" (p. 38).

When you interpret cardinality, you must consider the notation that appears at both ends of the relationship.

Possible end labels are

- 0..1 (zero or one match)
- 1..1 (exactly one match)
- 0..n (zero or more matches)
- 1..n (one or more matches)

The first part of the notation specifies the type of join for this relationship:

- an inner join (1)

  An inner join shows all matching rows from both objects.

- an outer join (0)

  An outer join shows everything from both objects, including the items that do not match. An outer join can be qualified as full, left, or right. Left and right outer joins take everything from the left or right side of the relationship respectively and only what matches from the other side.

Your users see a different report depending on whether you use an inner or outer join. For example, your users want a report that lists salespeople and orders. If you use an outer join to connect salespeople and orders, the report shows all salespeople, regardless of whether they have any orders. If you use an inner join, the report shows only the salespeople who have placed orders.

Data in one object might have no match in the other object. However, if the relationship has a minimum cardinality of 1, an inner join is always used and these records are ignored. Conversely, if all the items match but the relationship in the model has a minimum cardinality of 0, an outer join is always used, although the results are the same with an inner join. For example, the underlying table for one object contains a mandatory (non-NULLable) foreign key for the other. Ensure that the data and cardinalities match.

The second part of the notation defines the relationship of query items between the objects.

Cardinality in Generated Queries

Framework Manager supports both minimum-maximum cardinality and optional cardinality.

In 0:1, 0 is the minimum cardinality, 1 is the maximum cardinality.

In 1:n , 1 is the minimum cardinality, n is the maximum cardinality.

A relationship with cardinality specified as 1:1 to 1:n is commonly referred to as 1 to n when focusing on the maximum cardinalities.

A minimum cardinality of 0 indicates that the relationship is optional. You specify a minimum cardinality of 0 if you want the query to retain the information on the other side of the relationship in the absence of a match. For example, a relationship between customer and actual sales may be specified as 1:1 to 0:n. This indicates that reports will show the requested customer information even though there may not be any sales data present.

Therefore a 1 to n relationship can also be specified as 0:1 to 0:n, 0:1 to 1:n, or 1:1 to 0:n.

Use the **Relationship impact** statement in the **Relationship Definition** dialog box to help you understand cardinality. For example, Sales Staff (1:1) is joined to Orders (0:n).

It is important to ensure that the cardinality is correctly captured in the model because it determines the detection of fact query subjects and it is used to avoid double-counting factual data.

When generating queries, Cognos 8 follows these basic rules to apply cardinality:

- Cardinality is applied in the context of a query.
- `1:n` cardinality implies fact data on the `n` side and implies dimension data on the `1` side.
- A query subject may behave as a fact query subject or as a dimensional query subject, depending on the relationships that are required to answer a particular query.

Sparse Data

When modeling for analysis or reporting, it is important to consider the nature of the business questions versus the nature of the data source.

A common scenario is that a relationship between a dimension and a fact table in a star schema is optional. This means that not every dimensional member is mandatory in the fact table. OLAP engines compensate for this by inserting an appropriate value when creating the OLAP structure for any dimensional intersection points that do not have data.

For example, an Analysis Studio user wants to create this report:

	2005	2006
Canada		1,000,000
Mexico	500,000	750,000
United States	1,000,000	1,250,000

When modeling, it is common to override optional relationships between dimensions and facts for improved performance. However, when performing analysis or reporting on sparse data where you require information about dimensional members that have no facts, outer joins must be enabled to ensure that data is returned for valid dimensional intersection points.

To enable outer joins, we recommend that you do the following:

- Check with your database administrator to ensure that the data source can support full outer joins.
- Import metadata with outer joins enabled.

Modify a Relationship

After you import data (p. 47) or create a relationship (p. 117) in Framework Manager, you can rename the relationship and redefine cardinality.

You can create custom relationship expressions by selecting an operator from the list or by manually changing the expression in the expression editor.

You can also create a complex expression for the relationship (p. 84).

You can view the relationships that already exist for an object by selecting the object and clicking **Launch Context Explorer** from the **Tools** menu.

Steps

1. Click a relationship and, from the **Actions** menu, click **Edit Definition**.
2. To modify existing elements, on the **Relationship Expression** tab, select the query items, cardinalities, and operator you want.

   The query items must have the same data type.
3. To create an additional join, on the **Relationship Expression** tab, click **New Link**, and define the new relationship.
4. To test the relationship, on the **Relationship SQL** tab, identify the number of rows you want returned and click **Test**.
5. Click **OK**.

   If your metadata is from an OLAP data source, click **Close**.

Create a Complex Expression for a Relationship

You can create complex expressions for relationships by using functions, parameters, and objects from the model.

Steps

1. Click a relationship and, from the **Actions** menu, click **Edit Definition**.
2. On the **Relationship Expression** tab, click the ellipses (...) button next to the **Expression** box.
3. Define the expression.

   If you insert session parameters (p. 163) or prompts (p. 149) and you want to specify the values that they represent when you test the expression, click the **options** button .
4. Click **OK**.

Create a Relationship

You create a relationship to join logically related objects that your users want to combine in a single report. This is useful for relationships between objects that were not selected during metadata import, were not joined in the data source, or are from multiple sources.

You can directly create a relationship between the query items.

You can also create a complex expression for the relationship (p. 84).

You can also use Framework Manager to automatically generate relationships (p. 86) between objects based on selected criteria.

You can view the relationships that already exist for an object by selecting the object and clicking **Launch Context Explorer** from the **Tools** menu.

Steps

1. Ctrl+click one or two dimensions, query subjects, or query items.
2. From the **Actions** menu, click **Create**, **Relationship**.

   If this relationship is a valid target for a relationship shortcut, Framework Manager asks if you want to create a shortcut to that relationship. For more information, see "Create a Relationship Shortcut " (p. 85).
3. Click **OK**.

   The **Relationship Definition** dialog box appears. You can use this dialog box to modify the relationship (p. 83).

Create a Relationship Shortcut

A relationship shortcut is a pointer to an existing relationship. You can use relationship shortcuts to reuse the definition of an existing relationship. Any changes to the source relationship are automatically reflected in the shortcut. You can also use relationship shortcuts to resolve ambiguous relationships between query subjects "Resolving Ambiguous Relationships" (p. 338).

Framework Manager asks whether you want to create a relationship shortcut whenever you create a relationship and both of the following conditions apply:

- At least one end for the new relationship is a shortcut.
- A relationship exists between the original objects.

Steps

1. Ctrl+click the objects that you want to participate in the relationship shortcut.
2. From the **Actions** menu, click **Create**, **Relationship**.

   Framework Manager asks if you want to create a shortcut to that relationship.
3. Click **Yes**.

   A list appears of all relationships in which one end is a model object and the other end is either another model object or a shortcut to another model object.
4. To retrieve all relationships in which both ends can be either a model object or a shortcut to a model object, click **Find All**.
5. Click the relationship that you want to be the target of the relationship shortcut.
6. Click **OK**.

Detect and Generate Relationships

You can use Framework Manager to detect and generate relationships between two or more existing objects in your model. This is useful when you import metadata in stages, or when you want to change the criteria that apply to existing relationships, such as whether they include outer joins.

When importing star schema metadata, avoid generating relationships based on matching column or query item names unless you have naming conventions in place. Data warehouses often apply naming standards to columns, such as `surr_key` as the default column name for surrogate keys in dimensions. In this case, generating relationships that are based on matching column names generates inappropriate relationships between all dimension tables.

Steps

1. Ctrl+click two or more objects.
2. From the **Tools** menu, click **Detect Relationships**.
3. Select the rules you want to apply to each pair of tables.

Rule	Result
Use primary and foreign keys	Creates joins that are based on primary key and foreign key relationships. The query item names do not have to match.
Use matching query item names that represent uniquely indexed columns	Creates joins between query items whose names and data types match, if one or both of the underlying columns are uniquely indexed.
Use matching query item names	Creates joins between query items whose names and data types match. This generates as many relationships as possible.

4. Indicate whether you want Framework Manager to detect and generate relationships between
   - the selected objects
   - each selected object and every object in the project that is not selected
   - the selected objects and every other object in the project
5. Identify whether you want Framework Manager to create outer joins or inner joins based on outer joins that exist in the data source.
6. If you want to disable the automatic prevention of double-counting, convert all `n` cardinalities to `1` by clearing the **Fact detection enabled** check box.

   For more information, see "Guidelines for Modeling Metadata" (p. 321).
7. Click **OK**.

Query Subjects

Information about query subjects for SAP BW metadata appears in a different topic (p. 211).

A query subject is a set of query items that have an inherent relationship.

You use Framework Manager to modify query subjects to optimize and customize the data that they retrieve. For example, you can add filters or calculations. When you change the definition of a query subject, Framework Manager regenerates the associated query items, ensuring that any changes to query subject properties are reflected in all query items for that query subject.

There are different types of query subjects in Framework Manager:

- data source query subjects (p. 87)
- model query subjects (p. 89)
- stored procedure query subjects (p. 90)

You may also be interested in this topic, "Query Subjects vs. Dimensions" (p. 333).

Data Source Query Subjects

Query subjects that are based on relational metadata are defined by SQL statements that describe how to retrieve data from the data source.

Data source query subjects directly reference data in a single data source. Framework Manager automatically creates a data source query subject for each table and view that you import into your model.

For example, you import the Employee detail fact table from the Great Outdoors Warehouse sample database. Framework Manager then creates a query subject using the following SQL statement:

```
Select * from [go_data_warehouse].EMPLOYEE_DETAIL_FACT
```

Framework Manager generates query subjects that represent tabular data from the data source. For example, a query subject that references an entire table contains query items that represent each column in the table. If the SQL selects only specific columns, only those columns are represented as query items.

Each data source query subject can reference data from only one data source at a time. However, the advantage of data source query subjects is that you can directly edit the SQL that defines the data to be retrieved. This means that you can insert parameters that control the data that the query retrieves and create query subjects based on arbitrary SQL.

You may have created SQL statements for query subjects without enclosing references to columns and tables in quotes. This works in relational data source environments that do not use a case-sensitive lookup against the metadata tables they parse. The SQL statement for the query subject may be Cognos SQL (p. 111), native SQL (p. 112), or pass-through SQL (p. 113). Native and pass-through SQL statements must be completely self-contained and must not reference anything outside that SQL, such as database prompts, variables, or native formatting that would normally be supplied by the calling application. Cognos SQL statements, however, are analyzed using metadata from either the model or the relational data source. By default, Cognos SQL is case-sensitive, so it looks up metadata using the names as they appear in the SQL statement.

To use multiple data sources for a query subject, use a model query subject that accesses the data source query subjects or other model query subjects.

Create a Data Source Query Subject

Data source query subjects directly reference data in a single data source. Framework Manager automatically creates a data source query subject for each table and view that you import into your model. You can create additional data source query subjects.

You can add any combination of objects to a query subject, such as query items, other query subjects, or dimensions. You can add stand-alone calculations and filters, and you can also embed calculations and filters in the query subject.

You can also create model query subjects, which are based on metadata that exists in your model (p. 89), and stored procedure query subjects, which are generated from the stored procedures in a relational data source (p. 90).

Steps

1. Select the namespace folder and, from the **Actions** menu, click **Create**, **Query Subject**.
2. In the **Name** box, type a name for the new query subject.
3. Click **Data Source**, and click **OK**.
4. Complete all the steps in the **New Query Subject** wizard.

   To ensure that the data source is uniquely identified for a data source query subject, do not exit the wizard before the **Finish** button appears.
5. Click **Finish**.
6. Right-click the query subject you created and click **Edit Definition**.
7. Click the **SQL** tab, and from the **Available database objects** box, drag objects to the **SQL** box.

   You can also insert a data source reference, insert a macro, embed a calculation (p. 153), and embed a filter (p. 156).
8. Choose the action you want:

Goal	Action
Provide control over granularity	Click the **Determinants** tab (p. 94).
Test the query subject	Click the **Test** tab (p. 103).
View the SQL	Click the **Query Information** tab (p. 108).
View the system tables from the data source	Select the **Show System Objects** check box.

9. Click **OK**.

   A warning appears if any modifications invalidated relationships, other query subjects, calculations, or filters.

10. Ensure that the **Usage** and **Regular Aggregate** properties are set correctly (p. 141).

You can now organize objects in the **Diagram** tab. **Tip:** From the **Diagram** menu, click **Auto Layout**.

You can also control the model area that is visible in the diagram. **Tip:** Click and hold the **overview** button in the bottom right corner and drag the pointer over the diagram .

Model Query Subjects

Information about model query subjects for SAP BW metadata appears in a different topic (p. 211)

Model query subjects are not generated directly from a data source but are based on query items in other query subjects or dimensions, including other model query subjects. By using model query subjects, you can create a more abstract, business-oriented view of a data source.

Usually, model query subjects are created in the business view, not the import view. For information about the business view, see "Organizing the Model" (p. 175).

Because model query subjects are based on the metadata in your model, they let you

- reuse complex SQL statements that exist in the model
- reference objects from different data sources in the same query subject

If you import a model query subject from another model, the model query subject will not work unless you also import the data source query subjects that the model query subject references.

If you want to edit the SQL, you must convert the model query subject into a data source query subject (p. 108).

Create a Model Query Subject for Relational Metadata

Model query subjects are based on query items in other query subjects or dimensions, including other model query subjects. By using model query subjects, you can create a more abstract, business-oriented view of a data source.

You can add any combination of objects to a query subject, such as query items, other query subjects, or dimensions. You can add stand-alone calculations and filters, and you can also embed calculations and filters in the query subject.

You can create a new model query subject by merging existing query subjects and query items (p. 98). You can also create data source query subjects, which directly reference data in a single data source (p. 87), and stored procedure query subjects, which are generated from the stored procedures in a relational data source (p. 90).

When you use a model query subject in the Cognos 8 studios, Cognos 8 uses the relationships you have defined for the model query subject, not the relationships defined for the underlying data source query subjects. If you want to ensure that the relationships defined for the underlying data

source query subjects are used, do not define relationships or determinants for the model query subject.

Steps

1. Select the namespace folder and, from the **Actions** menu, click **Create**, **Query Subject**.
2. In the **Name** box, type a name for the new query subject.
3. Click **Model**, and click **OK**.
4. To add items to the model query subject, click the **Query Subject Definition** tab, and from the **Available Model Objects** box, drag objects to the **Query Items and Calculations** box.

   You can also embed a filter.

   If you add calculations (p. 153) or filters (p. 156) to a model query subject, Cognos 8 must go to the data source instead of simply accessing the model.

   You can change the order of items and calculations for a model query subject. However, if the query subject contains a query item folder, you can change the order only in the **Project Viewer**.
5. If you want to test the query subject, click the **Test** tab (p. 103).
6. Click **OK**.

   A warning appears if any modifications invalidated relationships, other query subjects, calculations, or filters.
7. Ensure that the **Usage** and **Regular Aggregate** properties are set correctly (p. 141).

You can now organize objects in the **Diagram** tab. **Tip:** From the **Diagram** menu, click **Auto Layout**.

You can also control the model area that is visible in the diagram. **Tip:** Click and hold the **overview** button in the bottom right corner and drag the pointer over the diagram .

Stored Procedure Query Subjects

Stored procedure query subjects are generated when you import a procedure from a relational data source. Framework Manager supports only user-defined stored procedures. System stored procedures are not supported.

The procedure must be run in Framework Manager to get a description of the result set that the procedure may return.

The stored procedure must return a single uniform result set. Cognos 8 supports only the first result set that is returned. If the procedure could conditionally return a different result set, the format must be consistent with the one used to define the metadata in Framework Manager.

Each result set must return the same form, such as the same number, types, and names of columns. Overloaded signatures are supported by Cognos 8, but each procedure must be defined as a uniquely named procedure with a separate query subject for each result set.

Output parameters are not supported.

After you import or create a stored procedure query subject, it appears as a broken object. You must run it to validate the underlying stored procedure and specify the projection list. Static metadata often does not exist for the stored procedure in the relational data source that describes what a result set may look like. The result set may be known only at run time. When a stored procedure is updated in the data source, running the stored procedure in Framework Manager updates the query subject using the newly generated query items.

Sometimes functions are imported as stored procedure query subjects. Review the stored procedure definition to determine what the procedure expects to be passed and what it attempts to return. Edit and test each stored procedure query subject that you think could be a function. If the test fails, the query subject is a function and must be deleted.

For more information, see "Create or Modify a Stored Procedure Query Subject" (p. 91).

Stored Procedures from Informix Data Sources

If you have stored procedures from Informix Dynamic or Parallel Server data sources, you must edit the parameters. We recommend that you refer to the original source of the stored procedures to ensure that they are mapped correctly.

Informix 7.x and 8.x provide only the name of the stored procedure to Framework Manager. You must provide all parameters, such as the parameter name, data type, mode, size, precision, scale, and value so that a result set can be obtained. Informix 9.x provides metadata for stored procedures and user defined functions with default parameter values. Check all parameters before using them, especially the mode attribute.

Informix functions are imported as stored procedures. After you import them, you must change them to functions by clicking the **f(x)** button in the **Edit Definition** dialog box. This button is enabled only for these functions. Then select the argument that represents the results or use the values obtained from the test results.

Stored Procedures from Composite Information Server

If you have stored procedures from Composite Information server, we recommend that you refer to the original source of the stored procedures to ensure that they are mapped correctly.

Composite functions are imported as stored procedures. After you import them, you must change them to functions by clicking the **f(x)** button in the **Edit Definition** dialog box. This button is enabled only for these functions. Then select the argument that represents the results or use the values obtained from the test results.

Create or Modify a Stored Procedure Query Subject

After you import or create a stored procedure query subject, you can modify it. To avoid inconsistencies, the modified query subject should return the same result set structure as the original stored procedure.

Framework Manager supports only user-defined stored procedures. System stored procedures are not supported.

There are different types of stored procedures:

Type of Stored Procedure	Description
Data Query	Issues a read-only transaction If you have a stored procedure with its type set to **Data Query**, the stored procedure issues a read-only transaction. When you run the stored procedure in Event Studio, an error message says that the stored procedure wants to update the database. The reason for the error is that the stored procedure contains a passive transaction that is supported by the underlying database. The solution is to click **OK** so that the stored procedure updates the database. No other action is required.
Data Modification	Writes a record to the data source. Use this type when you want to use the stored procedure in Event Studio. If you want Event Studio users to be able to select a parameter in a task, you must put quotation marks around the parameter. **Warning:** Testing a data modification stored procedure in the **Edit Definition** dialog box results in data being written to the data source. You cannot roll back transactions to the data source in Framework Manager. If undesired data is written to the data source as a result of testing the stored procedure, a rollback can be done by the database administrator if the data source is configured to support it. To test the stored procedure without data being written to the data source, click **Test** from the **Tools** menu.

You can also create data source query subjects, which directly reference data in a single data source (p. 87), and model query subjects, which are based on metadata that exists in your model (p. 89).

Steps

1. Do the following:

Goal	Action
Create a stored procedure query subject	• Select the namespace folder and, from the **Actions** menu, click **Create**, **Query Subject**. • In the **Name** box, type a name for the new query subject. • Click **Stored Procedure**, and click **OK**. • Complete all the steps in the **New Query Subject** wizard.

Goal	Action
Modify a stored procedure query subject	• Select the stored procedure query subject that you want to modify. • From the **Actions** menu, click **Edit Definition**.

2. Click the **Definition** tab and choose the action that you want.

Goal	Action
Use a different stored procedure	In the **Stored Procedure Name** box, type the name of the stored procedure.
Change the type of the stored procedure	From the **Type** box, select **Data Query** or **Data Modification**.
Change which data source the stored procedure is in	Click the ellipsis (...) button next to the **Data Source** box. When you import a stored procedure, a new data source is created. You can point to the original data source and delete the new one.
Edit an argument	Click the argument and click the ellipsis (...) button. The **Syntax** box in the **Query Subject Definition** dialog box shows the correct syntax to use.
Generate the projected query items	Click the **Test** tab (p. 103).

3. Click **OK**.

   Framework Manager runs the stored procedure and, if the query subject returns a result set, validates the query subject.

   If the stored procedure does not return a result set, the query subject becomes an invalid query subject if saved in the model. If the invalid query subject is included in the published package, the invalid query subject cannot be used in a report.

4. Ensure that the **Usage** and **Regular Aggregate** properties are set correctly for each newly created query item.

   For example, a query item may be set as a fact when it is an identifier.

You can update the stored procedure query subject if the data source changes (p. 107).

Example - Use Prompts with a Stored Procedure

If you define prompts for stored procedure variables, your users can set the variables in reports.

Steps

1. Create a stored procedure query subject that uses the sp_FIND_ORDER_DATE stored procedure.

   The **Query Subject Definition** dialog box appears.

2. On the **Definition** tab, select the `@order_number` argument, and click the ellipsis (...) button.

3. In the **Value** box, type the following macro syntax and then click **OK**:

   ```
   #prompt('Order Number','integer')#
   ```

   Note: Framework Manager removes anything that is outside the number signs when running the macro.

4. If you want to test the prompt for the variable, do the following:

   - Click the **Test** tab, and then click **Test Sample**.

     The **Prompt Values** dialog box appears.

   - In the **Name** column, click Order Number.

   - In the **Value** field, type **1234** and click **OK**.

     One record is returned, showing the date for Order Number 1234.

     Framework Manager uses this value for the duration of the current session or until you clear the prompt value.

5. Click **OK**.

Determinants

Determinants reflect granularity by representing subsets or groups of data in a query subject and are used to ensure correct aggregation of this repeated data. Determinants are most closely related to the concept of keys and indexes in the data source and are imported based on unique key and index information in the data source. We recommend that you always review the determinants that are imported and, if necessary, modify them or create additional ones. By modifying determinants, you can override the index and key information in your data source, replacing it with information that is better aligned with your reporting and analysis needs. By adding determinants, you can represent groups of repeated data that are relevant for your application.

An example of a unique determinant is Day in the Time example below. An example of a non-unique determinant is Month; the key in Month is repeated for the number of days in a particular month. When you define a non-unique determinant, you should specify group by. This indicates to Cognos 8 that when the keys or attributes associated with that determinant are repeated in the data, it should apply aggregate functions and grouping to avoid double-counting. It is not recommended that you specify determinants that have both uniquely identified and group by selected or have neither selected.

Year Key	Month Key	Month Name	Day Key	Day Name
2006	200601	January 06	20060101	Sunday, January 1, 2006
2006	200601	January 06	20060102	Monday, January 2, 2006

You can define three determinants for this data set as follows -- two group by determinants (Year and Month) and one unique determinant (Day). The concept is similar but not identical to the concept of levels and hierarchies.

Name of the Determinant	Key	Attributes	Uniquely Identified	Group By
Year	Year Key	None	No	Yes
Month	Month Key	Month Name	No	Yes
Day	Day Key	Day Name Month Key Month Name Year Key	Yes	No

In this case, we use only one key for each determinant because each key contains enough information to identify a group within the data. Often Month is a challenge if the key does not contain enough information to clarify which year the month belongs to. In this case, however, the Month key includes the Year key and so, by itself, is enough to identify months as a sub-grouping of years.

Note: While you can create a determinant that groups months without the context of years, this is a less common choice for reporting because all data for February of any year would be grouped together instead of all data for February 2006 being grouped together.

When to Use Determinants

While determinants can be used to solve a variety of problems related to data granularity, you should always use them in the following primary cases:

- A query subject that behaves as a dimension has multiple levels of granularity and will be joined on different sets of keys to fact data.

  For example, Time has multiple levels, and it is joined to Inventory on the Month Key and to Sales on the Day Key. For more information, see "Multiple-fact, Multiple-grain Queries" (p. 327).

- There is a need to count or perform other aggregate functions on a key or attribute that is repeated.

  For example, Time has a Month Key and an attribute, Days in the month, that is repeated for each day. If you want to use Days in the month in a report, you do not want the sum of Days in the month for each day in the month. Instead, you want the unique value of Days in the

month for the chosen Month Key. In SQL, that is `XMIN(Days in the month for Month_Key)` . There is also a `Group by` clause in the Cognos SQL.

There are less common cases when you need to use determinants:

- You want to uniquely identify the row of data when retrieving text blob data from the data source.

  Querying blobs requires additional key or index type information. If this information is not present in the data source, you can add it using determinants. Override the determinants imported from the data source that conflict with relationships created for reporting.

  You cannot use multiple-segment keys when the query subject accesses blob data. With summary queries, blob data must be retrieved separately from the summary portion of the query. To do this, you need a key that uniquely identifies the row and the key must not have multiple segments.

- A join is specified that uses fewer keys than a unique determinant that is specified for a query subject.

  If your join is built on fewer columns than what is stored in Framework Manager within the determinants, there will be a conflict. Resolve this conflict by modifying the relationship to fully agree with the determinant or by modifying the determinant to support the relationship.

- You want to override the determinants imported from the data source that conflict with relationships created for reporting.

  For example, there are determinants on two query subjects for multiple columns but the relationship between the query subjects uses only a subset of these columns. Modify the determinant information of the query subject if it is not appropriate to use the additional columns in the relationship.

Specify a Determinant

Determinants provide control over granularity for query subjects.

If a query subject has determinants, each query item of the query subject must be included in one of the determinants.

Determinants are processed in the order in which they are specified in the model. You can change the order of the determinants. If a query subject has more than one determinant, the first one that covers all the requested items is used. Determinants are evaluated in the context of each required join as well as the context of requested items.

Data source query subjects are imported with determinants defined for them. These default determinants are generated based on keys and indexes in the data source.

Constraints

Model query subjects do not have determinants defined for them automatically. If determinants are needed, you must define them manually.

Stored procedure query subjects do not have determinants.

You cannot use determinants with user-entered SQL that was specified in a query defined in Report Studio.

Steps

1. Click the query subject you want and, from the **Actions** menu, click **Edit Definition**.
2. Click the **Determinants** tab.
3. To add a query item to a determinant, select the query item from the **Available items** box and click **Add**.
4. To specify that the selected query item is the lowest level of the hierarchy and should be used as the unique identifier, select the **Uniquely Identified** check box.

   Do this only if the data in this item is unique for every row in the underlying data source.

   If you define a determinant as unique and you select the **Group By** check box, grouping is applied after the determinants are processed. The `Group By` clause is used in the generated SQL, which enables aggregate functions in the data source to be used in Cognos 8.

   If you have multiple determinants with **Uniquely Identified** check box selected, only the **Group By** setting of the first determinant is used. This is because the processing of determinants stops at the first determinant with **Uniquely Identified** check box selected.
5. To indicate that, when the keys or attributes associated with the determinant are repeated in the data, Cognos 8 should apply aggregate functions and grouping to avoid double-counting, select the **Group By** check box.

   It is not recommended that you specify determinants that have both uniquely identified and group by selected or have neither selected.
6. To define a key, right-click one or more query items in the **Available items** box and click **Add as Key**.

   You can also drag query items to the **Key** box.
7. To identify which query items users must report on, right-click query items in the **Available items** box and click **Add as Attributes**.

   You can also drag query items to the **Attributes** box.

   If a determinant defines attributes, all query items in the query subject must be associated with the determinant and defined as either a key or an attribute. You can have a determinant with no attributes defined for it. Framework Manager uses this type of determinant to indicate which query items are indexed.
8. If you want to change the order of the determinants, use the arrow buttons.

   Determinants are processed in the order in which they are specified in the model.
9. Click **OK**.

For more information, see "Determinants" (p. 324) and "The SQL Generated by Cognos 8 " (p. 351).

The Effect of Determinants on SQL

It is important to understand the effect that determinants have on the SQL that is generated. Determinants affect the grouping and aggregation of data, including other query subjects that have relationships with the query subject as well as the query subject itself.

For example, consider the following information. Each Product Line contains many occurrences of Product Type. Each Product Type contains many occurrences of Product. For Product, Product Key is a surrogate key and Product Number is a business key that is used as an attribute. Data joined on Product Key is aggregated correctly when reported by Product Line or Product Type or both.

Determinant	Key	Group By	Uniquely Identified	Attributes
Product line	Product line code	Yes		Product line
Product type	Product type code	Yes		Product type
Product	Product key		Yes	Cost Margin Product name Product number

Create a Model Query Subject Based on Existing Objects

Information about SAP BW model query subjects appears in a different topic (p. 212).

You can select existing model objects and merge them into a new model query subject. This means that you can reuse existing metadata to quickly create query subjects.

The objects that you can merge include

- relational data source query subjects and their shortcuts
- model query subjects and their shortcuts
- query items, filters, and calculations in model and data source query subjects
- relationships and relationship shortcuts between model and data source query subjects

You can merge any number of the same type of objects into a new query in a single operation. The merge always creates a new model query subject.

The new query subject contains any filters that exist in the original query subject.

Steps

1. Ctrl+click the objects that you want to merge into a single query subject.

2. From the **Actions** menu, click **Merge in New Query Subject**.

View Related Objects

You can explore a visual representation of the objects that are connected to the query subject or dimension that you select in the **Project Viewer**. The **Context Explorer** shows the objects that the selected object is connected to. You can also select a connected object and see its references.

You can hide an object in the **Context Explorer**. You can also change the layout, fit all objects in the **Context Explorer**, and zoom in and out.

You can also use the **Dimension Map** tab to explore dimensions.

Steps

1. Select one or more objects that you want to explore.
2. From the **Tools** menu, click **Launch Context Explorer**.
3. To see the connected objects, click one or more objects and click the appropriate button.

Goal	Button
View the objects that are related to the selected object.	
View the immediate references for the objects.	
View all references for the objects.	

4. If you want to see details about an object, such as its relationships and query items, right-click the object, click **Diagram Settings**, and then select the details you want.

Create a Query Set

A query subject can be defined using the `set` operations of union, intersect, or except. You define a query set to merge, compare, or equate similar data from different data sources. Query sets are useful when modeling data from disparate systems.

There are many reasons for creating a query set. A query set may be needed to create a conformed dimension across disparate data sources, such as relational and SAP BW data sources. Or, you may want to compare the contents of two queries to determine whether the queries contain the same data; this is common in test environments. Or, you may want to compare queries that return nulls. Or, you want to handle a fact-to-fact relationship that is truly a one-to-one relationship. (If it is not truly a one-to-one relationship, create a multiple-grain query (p. 327).)

A query set can consist of only two query subjects. You can create a query set that merges two other query sets together. A query set can contain

- all the rows of two query subjects (union operation)

For example, your company recently acquired another company and you need a complete list of all customers.

- only the rows that are shared between the query subjects (intersect operation)

  For example, you want to find out which staff members are also managers.

- only the rows that exist in the first query subject and not in the second query subject in the query set (except operation)

  For example, you want to highlight the differences between where your products were sold this year and ten years ago.

The names of the items in the projection list default to the items assigned to the first query subject in the set operation.

Relationships between the two query subjects in the query set and other query subjects are not included in the query set.

Reports show different results depending on which operator is used. For example, you have two query subjects with the names of various employees.

The first query subject contains these rows:

Jane
John
John
Michael
Michael

The second query subject contains these rows:

Jane
John
John
Patrick

You create a query set. You see different results depending on the operator you use.

Operator	Result	Notes
Union	Jane, John, Michael, Patrick	All items are shown. Values are not duplicated.
Intersect	Jane, John	Items in common are shown. Values are not duplicated.
Except	Michael	Items that are not common are shown. Values are not duplicated. If the second query subject were listed first in the query set, the result would be Patrick.
Union All	Jane, Jane, John, John, John, John, Michael, Michael, Patrick	All items are shown. Values are duplicated.
Intersect All	Jane, John, John	Items in common are shown. Values are duplicated.
Except All	Michael, Michael	Items that are not common are shown. Values are duplicated. If the second query subject were listed first in the query set, the result would be Patrick.

Constraint

Not all data types are supported. Generally, sets are not permitted on `BFILE`, `BLOB`, `CLOB`, `LONG`, and `VARRAY` data types, or on nested table columns.

Steps

1. Select two query subjects that meet these requirements:
   - Each query subject must have the same number of columns.
   - Columns must be in the same order.
   - Columns must have the same or similar data types.

     The data types do not need to be exactly the same if those in the second result set can be automatically converted by the data source to data types compatible with those in the first result set.

     For example, one query subject contains country data and uses `int` as the data type. Another query subject contains country data and uses `smallint` as the data type. Framework Manager imports these query subjects as `int16` and `int32` and performs a `set` operation.

2. From the **Actions** menu, click **Define Query Set**.
3. Click the **Definition** tab.
4. In the **Name** box, give the query set a name.
5. Review the **Query Subject** boxes to ensure the order that the query subjects will appear in the `Select` clause is correct.

   The order could be important if you want a specific set of column names (aliases) that appears in only one of the query subjects. If the order is incorrect, cancel this query set and start again.

   For union and intersect, the order of the query subjects does not matter. You can change the order and receive the same answer. For except, the order of the query subjects does matter.
6. Use the **Operator** box to define how the rows of the query subjects are combined.

Option	Description
Union	Retrieves all unique rows from both sets. Duplicates are removed.
Intersect	Retrieves rows that are common between the query subjects.
Except	Retrieves rows that exist in the first query subject and not in the second query subject.

7. To create a Union All, Intersect All, or Except All operation, clear the **Remove Duplicate Row** check box.
8. Choose the action that you want.

Goal	Action
Work with the calculations that are embedded in the query subjects	Click the **Calculations** tab. You can add or edit the calculations and change the order of the calculations.
Work with the filters that are embedded in the query subjects	Click the **Filters** tab. You can add or edit the filters, change the order of the filters, and change the usage of filters.
Test the query set	Click the **Test** tab.

9. Click **OK**.

You may be interested in the following related topics:

- embedded calculations (p. 153)

- embedded filters (p. 156)
- determinants (p. 94)
- testing the query set or changing the test settings (p. 103)

Test a Query Subject or Query Set

You can see the results that an object returns by testing it. You can test when creating an object or later on. The objects you can test are dimensions, query subjects, query sets, hierarchies, levels, calculations, and query items.

You can view the data that will appear in a specific report before publishing a package by selecting and testing the objects that will appear in the report. This makes it easier to debug a model and to verify that the model meets the reporting requirements because you do not need to create and publish packages first.

When you test an object, Framework Manager returns sample data. Formatting is not applied to the sample data. If you must test formatting, you must publish the package and view the objects in the Cognos 8 studios.

You may see different results depending on what you test. For example, if you use the expression editor to test a calculation that is embedded in a query subject, Framework Manager tests only the expression, not the item, so the aggregation setting for the query item is not applied to the test. Testing the entire query subject, which includes the calculation, gives a different result because the aggregation setting is applied. For example, if the aggregation setting is summarize, you can see a smaller number of rows in the test.

If you test a child segment of a segmented model, you may see an error if an object you are testing refers to an object in another child segment and the referenced object is not available to the project you are in. We recommend that you check that the parent model contains all the objects and that this error message does not appear when you test the parent model.

Governor settings may affect the testing results. For more information, see "Set Governors" (p. 310).

You can change existing test settings to customize the results that the test shows. For example, in addition to other settings, you can control the number of rows returned.

Steps When Creating or Modifying the Object

1. Select the object you want to test.
2. From the **Actions** menu, click **Edit Definition**, and click the **Test** or **Query Information** tab.

   The **Test Results** box is initially empty until you run the query.

   Any result sets that contain binary large objects are shown as [blob].
3. To run the query and bring back all the test results, click **Test Sample**.
4. If you want to add a count of the rows, click **Total Rows**.

5. If you want to apply the **Regular Aggregate** property of the query item or the **Aggregate Rules** property of a semi-additive measure that is referenced in the expression, select the **Auto Sum** check box.

   If you clear this check box, a row is returned for each row in the result set of the query.

6. If you want to obtain more information about the query results, click the **Query Information** tab.

7. Click **OK**.

Steps to View the Data That Will Appear in a Specific Report

1. Select the objects that will appear in the report.

2. From the **Tools** menu, click **Test**.

   A message appears, telling you if the test was successful or if problems were found.

3. To view details about any problem that is found, click the **Query Information** tab.

   If you do not see the results of the query in the test window, the data from your data source may exceed the value of the **Runtime Limits** governor. The query stops at the specified limit, but the test result window does not contain any data. **Tip:** Set the **Runtime Limits** governor to zero.

Change the Test Settings

You can customize the tests by changing the test settings.

Steps

1. Select the object that you want.

2. From the **Actions** menu, click **Edit Definition** and then click the **Test** tab or the **Query Information** tab.

3. Click **Options** and then click the **Test Settings** tab.

4. Choose the options that you want.

Goal	Action	Persistence
Retrieve all data and show a specified number of rows	Select the **Restrict the maximum number of rows to be returned** check box and type the required number of rows. This setting does not improve performance for retrieving data when testing dimensions, query subjects, and query sets.	This setting applies to all dimensions, query subjects, and query sets in the model. This setting is saved and used in your next session with any model.

Goal	Action	Persistence
Specify the level of detail	Drag the **Level of Information shown in Results Information** slider to the location that represents the amount of detail you require.	This setting is saved and used in your next session with this model.
Temporarily override session parameters	In the **Session Parameters** box, click **Set**. The **Session Parameters** dialog box appears.	The override values are not saved with the model. This setting is for your current session only.
Apply relevant design mode filters	Select the **Apply all relevant design mode filters when testing** check box. This applies all relevant filters whose usage is set to design mode in another dimension, query subject, or query set.	This setting is saved and used in your next session with any model.
Apply a security filter	In the **Security Filters** box, click **Edit**.	This setting is saved and used in your next session with this model.
Change the prompt values	In The **Current Prompt Values** box, click **Prompts**. The **Model Prompts Manager** dialog box appears, which shows all prompts, and their values, that are in the model.	The prompt values are not saved with the model. This setting is for your current session only.

5. Click **OK** two times.

You may be interested in the following related topics:

- setting governors (p. 310)
- security filters (p. 253)
- temporarily overriding session parameters (p. 163)
- changing prompt values (p. 149)
- working with dimensions (p. 115)
- working with query subjects (p. 87)

Validate a Query Subject

Information about validating SAP BW query subjects appears in a different topic (p. 213).

You can validate the definition of the query subject without having to open the **Query Subject Definition** dialog box. This is useful to do when

- new query items were added to a query subject
- the definition of the underlying query subject has changed
- the parameters of a stored procedure were changed

The **Evaluate Object** command completes an exhaustive check of all query subjects and ensures that they can run.

What happens in the evaluation process depends on the type of query subject selected.

Type of query subject	Evaluation process
Relational data source query subject	A request based on the derived items is sent to the relational data source. The list of data source references is updated. The physical attributes, such as data type, are updated as needed.
Model query subject based on relational metadata	A request based on the derived items is sent to the data source. The cached SQL, if available, is updated. The physical attributes, such as data type, are updated as needed.
Stored procedure query subject	A request based on the latest parameters of the stored procedure is sent to the data source. The list of derived query items is updated.

You can also update the query subject (p. 107) if it is from a relational data source, or synchronize the entire project (p. 306).

Steps

1. Select the query subject that you want to evaluate.
2. From the **Tools** menu, click **Evaluate Object**.

If you changed the **Regular Aggregate** property to **unsupported**, the property is reset when you evaluate the query subject. If the property is set to any other value, the property is not changed.

Update Query Subjects

If you are using a relational data source, you can choose to update only the query subjects instead of performing a full project synchronization. You must perform a project synchronization to synchronize changes in a third-party data source.

The query subject is updated based on the definition in the data source. When you update a query subject, new metadata is fetched from the data source and query items are re-synchronized.

You can also evaluate the query subject, if it is from a relational data source (p. 106).

Constraint

You cannot use the **Update Object** command for model query subjects.

Steps

1. Select one or more query subjects.
2. From the **Tools** menu, click **Update Object**.

   Tip: You can instead open the **Query Subject Definition** dialog box and click **OK**.

Convert a Query Subject into a Dimension

You can convert a query subject into a regular dimension or a measure dimension when you want to use features associated with dimensions, such as defining hierarchies and levels. A model query subject becomes a model dimension.

While you can convert data source query subjects to data source dimensions, data source dimensions have limited functionality in comparison to query subjects or model dimensions. It is recommended that you discontinue using data source dimensions (both regular and measure). Create new models following the recommendations in "Guidelines for Modeling Metadata" (p. 321) to use query subjects as the relational foundation of the model. Define regular and measure dimensions as model objects based on data source query subjects or model query subjects or both. Guidance on migration for existing users of data source dimensions will be provided in a future release.

If the query subject has determinants specified for it, the keys you specified for each determinant is used to build a hierarchy with a business key for each level. The determinants form one hierarchy. The first string attribute for each determinant is used for the business caption. If the caption is not the attribute you want to use, you must change it manually.

Converting a query subject into a dimension is simply a starting point. We recommend that you examine each dimension that is created this way to ensure that it reflects your requirements.

You can also convert a dimension into a query subject (p. 131).

Constraints

You cannot use determinants to create separate hierarchies for the dimension. You must create the separate hierarchies for the dimension after converting the query subject.

You cannot convert the following to dimensions:

- query sets
- stored procedure query subjects
- SAP BW query subjects

Steps

1. Select the query subjects that you want to convert.
2. From the **Actions** menu, click **Convert to Regular Dimension** or **Convert to Measure Dimension**.

Convert a Model Query Subject into a Data Source Query Subject

You can convert a model query subject into a data source query subject if you want to edit the SQL.

Constraint

Do not convert the model query subject if you want it to reference multiple data sources.

Steps

1. Select the model query subject that you want to convert.
2. From the **Actions** menu, click **Convert to Data Source Query Subject**.

   This command is available only if you have run the query and the **Query Information** tab in the **Edit Definition** dialog box contains SQL.

Edit the SQL

SQL is the industry-standard language for creating, updating, and querying relational database management systems. When you edit the definition of a relational data source query subject, you can use

- Cognos SQL (p. 111)
- native SQL (p. 112)
- pass-through SQL (p. 113)

If you want to edit the SQL of a model query subject, you must copy the SQL for the model query subject from the **Query Information** tab and paste it into a new data source query subject. You can also convert the model query subject into a data source query subject (p. 108). Do not edit the SQL if you want the model query subject to reference multiple data sources.

Changing the alias of a column regenerates the query item that represents that column. Any modifications that you made to the query item are not retained because Framework Manager considers it a new query item.

You can add comments to the SQL by using `/*` before the comment and `*/` at the end.

Here is an example:

```
select country /* this is a multiline
comment
another line
another line
*/
```

Steps

1. Click the data source query subject that you want to change.
2. From the **Actions** menu, click **Edit Definition**.
3. Click the **SQL** tab, and drag objects into the **SQL** box or type in the SQL you want.
4. Click **OK**.

Change the Type of SQL

When choosing the type of SQL in which to generate a data source query subject, you must weigh the following factors and decide which are most important.

Type	Advantage	Disadvantage
Cognos SQL	Cognos SQL improves query subject performance; for example, by removing unused elements at query time. SQL works on any supported database.	You cannot enter non-standard SQL.
Native SQL	Performance is optimized across all related query subjects. You can use SQL that is specific to your database.	You cannot use SQL that the data source does not support for subqueries. The SQL may not work on a different database type.
Pass- Through SQL	You can enter any SQL supported by the database.	There is no opportunity for Framework Manager to automatically optimize performance. The SQL may not work on a different data source.

Note that you cannot change the type of SQL for query subjects that are based on OLAP data sources.

Prerequisites for Changing to Native SQL

If you change an existing query subject to native SQL, you must first ensure that the SQL reflects the rules that apply to the native data source so that your query runs properly. You must do the following:

- Edit existing table names.

  Cognos SQL uses a two-part structure to name query subjects. For example, `[gosales].[ProductLine]` means that the ProductLine query subject comes from the gosales database. Therefore, when you switch to native SQL, you must ensure that all table names include the parent elements required by the data source vendor.

  For information about naming conventions, see "Naming Conventions for Objects in a Project" (p. 43).

- Ensure that the SQL is valid for subqueries.

  Framework Manager processes native SQL query subjects as subqueries. For example, here is a Cognos SQL query subject:

  ```
  Select
  P.ProductName, P.Margin From Product P
  ```

  If you change it to native SQL, Framework Manager generates the following statement:

  ```
  Select
  oracle_plain.ProductName as Productname,
  oracle_plain.Margin as Margin
  From
  (GOSALES1_OR_92_WE...SELECT
  P.PRODUCTNAME, P.MARGIN
  FROM
  PRODUCT P}
  )oracle_plain
  ```

  Therefore, you must ensure that the query subject adheres to additional database restrictions that are imposed on subqueries, such as not using the `With` clause. Pass-through SQL does not have the same restrictions. However, the fact that native SQL is processed as part of a larger query improves its performance.

  To test native SQL using a query tool, such as Oracle's SQL*Plus, you must place the SQL in the `From` clause of a `Select` statement. For example, you can use the following syntax in a query tool:

  ```
  Select * from (<Native SQL>) T1
  ```

- Assign aliases to derived columns.

  Assign alias names to any column whose values are calculated. Here is an example:

  ```
  SELECT
  Length(Country) as LGTH
  FROM Country
  ```

- Insert double quotation marks around alias names.

  Changing the SQL type of a query subject can change the case of alias names. When this happens, any query subject that references the changed query item becomes invalid. To ensure that there is no case change, insert double quotation marks around the alias, such as

  ```
  Select
  COUNTRY as "test" from COUNTRY
  ```

- If a data source query subject contains a macro in the projection list (`Select` clause) of the SQL statement, specify an alias in the SQL that matches the **Column Name** property of the query item.

  An error could occur because the macro evaluates to a column name that is different from the **Column Name** property of the corresponding query item. The result is that the system is unable to locate the item in the projection list. Projection lists are static.

  Assigning an alias ensures that the name of the item in the projection list remains constant, as the results of evaluating the macro change.

  For example, the following query contains a session parameter, runLocale, whose value specifies which column the query retrieves:

```
Select
#$ColumnMap{$runLocale}# as
CountryNameAlias
From
[GoSales].Country
```

  Note that the number sign (#) is reserved for macros. Framework Manager removes anything that is outside the number signs when running the macro.

Steps

1. Click the query subject that you want to change.
2. From the **Actions** menu, click **Edit Definition**, and then click the **Query Information** tab.

   The **Test Results** box is initially empty until you run the query.
3. Click **Options**, and then click the **SQL Settings** tab.
4. Use the **SQL Type** list to change the type of SQL.

   If you are changing the type to native SQL, see the checklist above to ensure that the SQL reflects the rules that apply to the native data source.
5. Click **OK**.
6. If you want to see the SQL, click **Test Sample**.
7. If you want to see the actual query, click **Query**.
8. If you want to see the xml that Cognos 8 uses, click **Response**.
9. Click **OK**.

Cognos SQL

By default, Framework Manager uses Cognos SQL to create and edit query subjects. Cognos SQL adheres to SQL standards and works with all relational and tabular data sources. Framework Manager generates the most optimized SQL possible. In this way, Cognos SQL is preferable.

Because query subjects in Framework Manager are similar to views in databases, the SQL for each query subject must conform to the SQL standards that apply to views. For example, you must assign aliases to any column that is empty or whose name is not unique. This level of conformance means

that Cognos SQL behaves more consistently than vendor-specific SQL, which does not adhere to SQL standards.

In general, using Cognos SQL is preferable because you can create query subjects that

- can contain metadata from multiple data sources
- have fewer database restrictions
- interact more effectively with Cognos applications

Constructs of the SQL Standard

If the data source supports it, you can use the `With` clause with Cognos SQL. The `With` clause is used to generate more readable SQL and to let the data source generate a more optimal plan for data retrieval. The data source can more easily detect the cases where the same tables must be scanned and can then resolve these as an inline view or temporary table.

By default, Framework Manager uses the common table constructor from the SQL standard when the **Use With clause when generating SQL** governor is set.

Use the `With` clause for better query performance if the request is restricted to functionality supported by the underlying data source software. When a request uses functionality that is not supported by the data source, using the `With` clause may cause additional decomposition of the query, which can lead to degraded performance. In this case, not using the `With` clause may generate a better set of queries to the underlying data source.

Here is an example of Cognos SQL using derived tables:

```
SELECT * FROM
(SELECT SNO C1, AVG(QTY) C2, COUNT(*) C3 FROM
SUPPLY
GROUP BY SNO) T1,
(SELECT MAX(QTY) C1 FROM SUPPLY) T2
```

The following shows how Cognos SQL turns the above example into a `With` clause:

```
WITH T1 AS (SELECT SNO C1, AVG(QTY) C2, COUNT(*)
C3 FROM
SUPPLY GROUP BY SNO),
T2 AS (SELECT MAX(QTY) C1 FROM SUPPLY)
SELECT *FROM T1, T2
```

Do not use the `With` clause for recursive processing.

For more information about the `With` clause, see "Set Governors" (p. 310).

Cognos continues to improve data type checking and SQL validation. Because of this and because not all vendors are completely compliant with the SQL standard, invalid or ambiguous SQL expressions that previously were passed to the data source will no longer be passed down. If you have an expression that returns a data type not specified by the SQL standard, we recommend that you pass the expression to the data source by using the syntax `{expr}`. Your users should use the same technique.

Native SQL

Native SQL is the SQL that the data source uses, such as Oracle SQL. Use Native SQL to pass the SQL statement that you enter to the database. Cognos 8 may add statements to what you enter.

You can not use native SQL in a query subject that references more than one data source in the project.

SQL specified in Framework Manager and processed by the database, whether native or pass-through, must be completely self-contained. It must not reference anything outside that SQL, such as database prompts, variables, or native formatting that would normally be supplied by the calling application.

If you are comfortable working with a native SQL version, you may want to use it for query subjects that are based on a single data source. By doing so, you can use keywords that are not available in Cognos SQL, and copy and paste SQL from another application into Framework Manager.

When the query is generated, Framework Manager combines the SQL of each query subject that uses a given data source connection into a single query. This helps improve the performance of the query. However, because the SQL is being generated as a series of subqueries, native SQL queries must adhere to any restrictions that their database vendor places on derived tables.

Here is an example of native SQL that returns a list of employees and managers:

```
SELECT
 lpad(' ', (level-1)* 4) ename
EMP_CHART,
 level, empno, ename, job, mgr
FROM
 emp
CONNECT BY PRIOR
 empno = mgr
AND
 deptno not in (20,30)
START WITH
 mgr IS NULL
ORDER BY
 level, job
```

Pass-through SQL

Use pass-through SQL when the SQL statement that you enter is not valid inside a derived table. Pass-through SQL lets you use native SQL without any of the restrictions that the data source imposes on subqueries. This is because pass-through SQL query subjects are not processed as subqueries. Instead, the SQL for each query subject is sent directly to the data source where the query results are generated.

Because each query subject is sent to the data source as a separate statement rather than being optimized by Framework Manager, performance is slower. Therefore, in choosing between native SQL and pass-through SQL, you must decide which is more important: performance or using SQL that is not permitted in a subquery.

Generally, you should use pass-through SQL only if you must create a query subject that contains constructs that are specific to a data source and that cannot be used inside a derived table, such as in a `With` or `OrderBy` clause.

SQL specified in Framework Manager and processed by the database, whether native or pass-through, must be completely self-contained. It must not reference anything outside of that SQL, such as database prompts, variables, or native formatting that would normally be supplied by the calling application.

For example, here is a systems-oriented report that contains the system date:

```
SELECT
```

```
 TO_CHAR(SYSDATE, 'DAY, DDTH MONTH YYYY')
FROM
 SYS.DUAL
```

Note that the number sign (#) is reserved for macros and that column names must be unique. Framework Manager removes anything that is outside the number signs when running the macro.

Change How the SQL Is Generated

You can specify how Framework Manager generates the SQL that retrieves data from relational data sources for data source query subjects (p. 87) or model query subjects (p. 89).

The **SQL Generation** type of a query subject can be set to either **As View** or **Minimized**. By default, it is set to **Minimized**.

When the generation type is set to **Minimized**, the generated SQL contains only the minimal set of tables and joins needed to obtain values for the selected query items.

When the generation type is set to **As View**, Framework Manager generates queries that contain the full SQL statement that defined the query subject. Use **As View** when you want to ensure that the query is run as a block. The SQL is treated as a view. For example, you want the query to return the same number of rows each time that it is run.

Using minimized SQL improves performance, resulting in a query that runs significantly faster. Generating minimized SQL is especially beneficial for query subjects that represent dimension tables. By using a single model query subject to model a dimension, you can benefit from small SQL queries that run significantly faster.

For example, the **SQL Generation Type** of the following query subject is **As View**. Note that this query subject contains a nested select statement.

```
select
 New_Query_Subject.COUNTRYCODE as COUNTRYCODE,
 New_Query_Subject.EUROINUSESINCE as
EUROINUSESINCE
from
 (select
 CONVERSIONRATE.COUNTRYCODE as COUNTRYCODE,
 COUNTRY.EUROINUSESINCE as EUROINUSESINCE
 from
 "2 - GOSales1 - OLE-DB".GOSALES1.dbo.CONVERSIONRATE
CONVERSIONRATE,
 2 - GOSales1 - OLE-DB".GOSALES1.dbo.COUNTRY COUNTRY
where
 (COUNTRY.SALESCOUNTRYCODE = CONVERSIONRATE.COUNTRYCODE)
 ) New_Query_Subject
```

If you change the **SQL Generation Type** to **Minimized**, Framework Manager generates the following simplified SQL:

```
select
 CONVERSIONRATE.COUNTRYCODE as COUNTRYCODE,
 COUNTRY.EUROINUSESINCE as EUROINUSESINCE
from
 "2 - GOSales1 - OLE-DB".GOSALES1.dbo.CONVERSIONRATE
CONVERSIONRATE,
 "2 - GOSales1 - OLE-DB".GOSALES1.dbo.COUNTRY COUNTRY
where
 (COUNTRY.SALESCOUNTRYCODE = CONVERSIONRATE.COUNTRYCODE)
```

Minimized SQL works best when the returned result sets of each query item are equivalent. If there are records in one column that do not correspond to records in another column, the result of the minimized query produces additional rows. You can avoid this by setting the **SQL Generation Type** to **As View**.

For example, if there are Product Types that are not used by any of the Products and these Product Types all have a common Product Line, a Product Line is reported for which there are Product Types, but for which there are no related Products.

Steps

1. Click the query subject that you want to change.
2. From the **Actions** menu, click **Edit Definition**, and then click the **Query Information** tab.

   The **Test Results** box is initially empty until you run the query.
3. Click **Options**, and then click the **SQL Settings** tab.
4. Set **Generate SQL** to **As View** or **Minimized**.
5. Click **OK**.
6. If you want to see the SQL, click **Test Sample**.
7. If you want to see the actual query, click **Query**.
8. If you want to see the xml that Cognos 8 uses, click **Response**.
9. Click **OK**.

Model Query Subjects and SQL Types

A model query subject that is based on another model query subject may use the logic of the parent query subject instead of its own logic. If the child model query subject uses the **Minimized** SQL type, it does not use the logic of the parent. If the child model query subject uses the **As View** SQL type, it uses the logic of the parent.

For example, you create a model query subject named Returned Products, which shows all return reasons for all products. When you run Returned Products, you see a list of over 700 items. You then create another model query subject based on Returned Products that is named Return Reasons. This model query subject contains only the Return Reason query item. If the SQL type is set to **Minimized**, the Return Reasons query subject shows five return reasons when it is run. If the SQL type is set to **As View**, the Return Reasons query subject uses the logic of the Returned Products query subject and shows over 700 items.

Dimensions

Information about dimensions based on SAP BW metadata appears in a different topic (p. 198).

A dimension is a broad grouping of data about a major aspect of a business, such as products, dates, or markets.

The types of dimensions that you can work with in Framework Manager are regular dimensions and measure dimensions. In SAP BW, measure dimensions are called key figures.

For example, in a project for sales analysis, you include these dimensions:

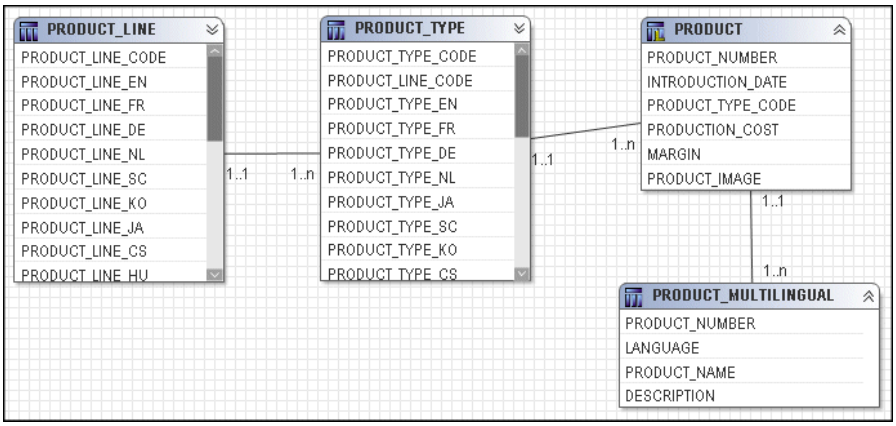
Name	Type	Description
Time	Regular dimension	Dates of sales organized into years, quarters, months, weeks, and days when sales were made
Region	Regular dimension	Locations of sales grouped into sales regions, countries, and cities
Product	Regular dimension	Product details organized by product type, brand, model, color, and packaging
Customer	Regular dimension	Customer information
Sales	Measure dimension	Purchase details such as units sold, revenue, and profit

You must use regular and measure dimensions to enable analysis on your relational data source. In most data sources, measure dimensions are likely to be shared by more than one regular dimension. Regular dimensions are often called shared dimensions. A measure dimension and regular dimensions organized into a cluster is often referred to as a star schema group but can also be referred to as a functional or subject area group.

You may also be interested in this topic, "Query Subjects vs. Dimensions" (p. 333).

Normalized Data Sources

Normalized or snowflaked data sources often have several tables that describe a single business concept. For example, a normalized representation of Product may include four tables related by `1..n` relationships. Each Product Line has one or more Product Types. Each Product Type has one or more Products. Products have names and descriptions in multiple languages so they exist in the Product Multilingual lookup table.



You may be interested in the following related topics:

- creating regular dimensions (p. 117)
- creating measure dimensions (p. 124)

Create a Regular Dimension

Information about modifying a regular dimension for SAP BW metadata appears in a different topic (p. 199).

A regular dimension contains descriptive and business key information and organizes the information in a hierarchy, from the highest level of granularity to the lowest. It usually has multiple levels and each level requires a key and a caption. If you do not have a single key for your level, it is recommended that you create one in a calculation.

Model regular dimensions are based on data source or model query subjects that are already defined in the model. You must define a business key and a string type caption for each level. When you verify the model, the absence of business keys and caption information is detected. Instead of joining model regular dimensions to measure dimensions, create joins on the underlying query subjects and create a scope relationship between the regular dimension and the measure dimension.

When creating regular dimensions, you must understand the dimensionality of the data. You must be able to answer the following questions:

- What are the levels in your dimension?
- What is the order and combination of levels that form hierarchies?
- What are the relationships between the levels?
- What uniquely identifies a level?
- Which data elements are associated at each level?
- Do you have more than one level of granularity, such as some data is recorded monthly and some is recorded daily?
- Are foreign keys defined in the data source?

You can specify multiple hierarchies on regular dimensions in Framework Manager. Multiple hierarchies for a regular dimension behave as views of the same query. However, you can use only one hierarchy at a time in a query. For example, you cannot use one hierarchy in the rows of a crosstab report and another hierarchy from the same dimension in the columns. If you need both hierarchies in the same report, you must create two dimensions, one for each hierarchy. For more information, see "Modeling Dimensions with Multiple Hierarchies" (p. 345).

In addition to creating regular dimensions, you can also merge dimensions into a single dimension or convert query subjects to dimensions.

Multiple-fact querying is enabled with conformed dimensions.

While you can use data source dimensions, they have limited functionality in comparison to query subjects or model dimensions. It is recommended that you discontinue using data source dimensions (both regular and measure). Create new models following the recommendations in "Guidelines for Modeling Metadata" (p. 321) to use query subjects as the relational foundation of the model. Define regular and measure dimensions as model objects based on data source query subjects or model query subjects or both. Guidance on migration for existing users of data source dimensions will be provided in a future release.

Steps

1. Select a namespace or folder where you want to place the dimension.
2. From the **Actions** menu, click **Create**, **Regular Dimension**, and then click the **Dimension** tab.
3. Click the **Dimension** tab.
4. Click **Add Hierarchy** and then drag one or more objects from the **Available items** box to the **Hierarchies** box.

   You can define multiple hierarchies for a dimension. The first hierarchy is used as the default, or primary, hierarchy.

   You can also create an alternate hierarchy by copying a level. Click a level and drag it to the right border of the dimension. You can only copy a level within the same dimension.
5. Click **Add Level** and then drag one or more objects from the **Available items** box into the new level.

   You can also create copies of levels in the **Dimension Definition** dialog box or in the **Dimension Map** tab. Click the level and drag it to another position in the hierarchy. All attributes of the level are also copied. You can only copy a level within the same dimension.
6. If you want to use a different item in a level, drag it from the **Available items** box to the **Select a level in the hierarchy control to see the query items** box.

   You are prompted to specify its role.

   By default, Framework Manager adds the name of the namespace.

   Tip: To have a multiple-part key such as first name plus last name, create a new attribute that combines the items, and then specify that the new attribute is the business key.

7. If you want to indicate that the keys of the levels above the current level are not necessary to identify the members in this level, select the item and select the **Unique Level** check box.

   Do this only if the key values are unique regardless of their context, such as postal code values.

8. Choose the additional tasks that you want to perform:
   - Specify roles (p. 123).
   - Embed calculations by clicking **Add** and then defining the expression (p. 153).

     To change a calculation that has been embedded in the dimension, in the **Dimension Map** tab, click **Attributes**, right-click the query item, and click **Edit Expression**.
   - Embed filters (p. 156).
   - Test the dimension (p. 128).
   - Edit the SQL and change various options (p. 111).
9. Click **OK**.
10. To change the default hierarchy for a dimension with multiple hierarchies, do the following:
    - In the **Properties** pane, click the ellipsis (...) button in the **Default Hierarchy** box.
    - Select a different hierarchy, and click **OK**.

You can also use the **Dimension Map** tab to create a regular dimension. Click the **regular dimension** button .

Hierarchies for a Regular Dimension

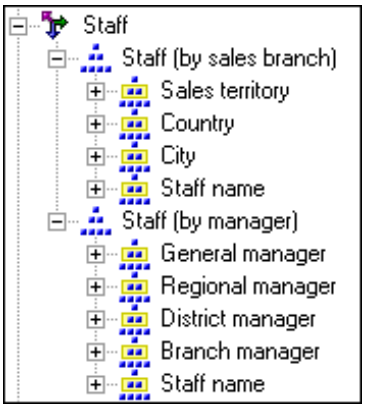
Information about hierarchies for SAP BW metadata appears in different topics (p. 200) and (p. 193).

A hierarchy is an ordered list of levels or a collection of items. Each query item in a hierarchy must have a unique name.

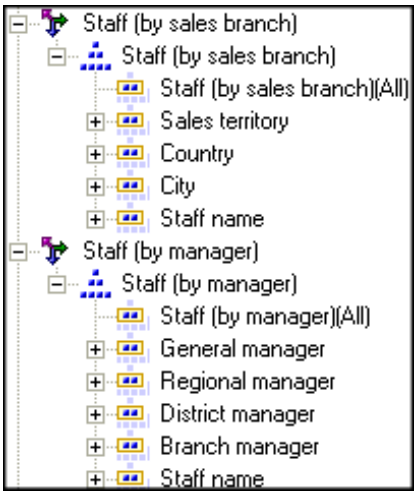
You can specify multiple hierarchies on regular dimensions in Framework Manager. Multiple hierarchies for a regular dimension behave as views of the same query. The first hierarchy is the primary or default hierarchy.

You can use only one hierarchy at a time in a query. For example, you cannot use one hierarchy in the rows of a crosstab report and another hierarchy from the same dimension in the columns. If you need both hierarchies in the same report, you must create two dimensions, one for each hierarchy. For more information, see "Modeling Dimensions with Multiple Hierarchies" (p. 345).

For example, sales staff can be viewed by manager or by sales branch and can be modeled as a single dimension with two hierarchies.



If you need both hierarchies in the same report query, such as on opposing axes, you must create a regular dimension for each hierarchy. For example, here is sales staff as two dimensions.



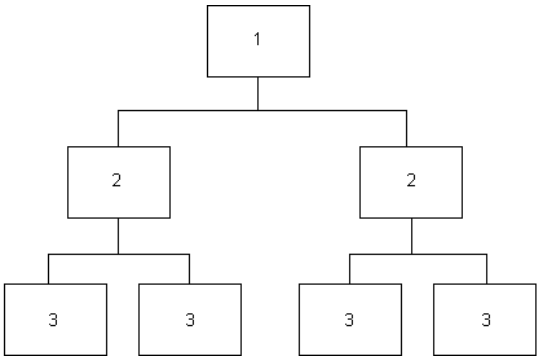
Tip: To change the default hierarchy for a dimension with multiple hierarchies, in the **Properties** pane, click the ellipsis (...) button in the **Default Hierarchy** box, and select a different hierarchy.

If a hierarchy in a dimension contains a large number of members, running a query in one of the Cognos 8 studios may be slow because the Cognos 8 engine is generating one large query for a locally-built cube. To resolve this issue, set the **Wide Member Tree** property in the **Properties** pane to **true**. The engine will then generate multiple smaller queries for the locally-built cube.

Balanced Hierarchy

Each path in a balanced hierarchy descends to the same depth.

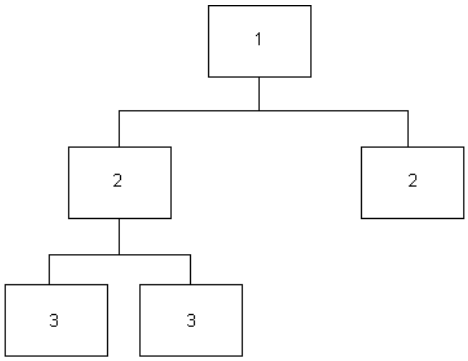
For example, in the following diagram, the highest level is Product Line. Level 2 is Product Type. Level 3 is Products.



Unbalanced Hierarchy

The branches in an unbalanced hierarchy descend to different levels.

For example, in the following diagram, the highest level in an organization is the CEO. Level 2 is the vice-presidents and the CEO's executive assistant. The executive assistant does not have subordinates although the vice-presidents do.



An unbalanced hierarchy can also be ragged. In a ragged-unbalanced hierarchy, there are gaps in the levels and the levels descend to different depths.

Ragged and Network Hierarchies

For relational metadata, we recommend that ragged hierarchies and network hierarchies be flattened in the data source.

Levels for a Regular Dimension

The simplest definition of a level consists of a business key and a caption, each of these referring to one query item. An instance (or row) of a level is defined as a member of that level. It is identified by a member unique name, which contains the values of the business keys of the current and higher levels. For example, `[gosales].[Products].[ProductsOrg].[Product]->[All Products].[1].[1].[2]` identifies a member that is on the fourth level, `Product`, of the hierarchy `ProductsOrg` of the dimension `[Products]` that is in the namespace `[gosales]`. The caption for this product is TrailChef Canteen, which is the name shown in the metadata tree and on the report.

The first level of the hierarchy is automatically defined as the All level. It contains a single root member, which represents the top level of the hierarchy. For example, the All level for the Time dimension is named Time (All). You cannot delete or move the All level. You can change its name, description, and screen tip.

If you do not specify the levels of the hierarchy correctly, incorrect aggregation could occur.

Member Unique Names

The member unique name (MUN) is how the member is found in the data source, much like using business keys to find records in a table.

The member unique name is used in the expression for a member data item that is used in a report, a reference to members in filters and expressions, and used in drill-through between OLAP data sources. The member keys in the MUN for the different OLAP data sources must match.

If a member unique name changes, members that are directly referenced in expressions, filters, or reports are no longer found because the MUN is contained in the report specification. Member unique names can change for a variety of reasons:

- Changes to the hierarchy and level structures may change the level unique name.
- The business key values have changed and this changed the member key path.
- The application changed during design or over time.
- The cube has category codes that are unpredictably unique.
- The production environment has more members than the test environment.
- The member no longer exists in the data source.

To avoid these problems, we recommend the following practices:

- Use unique codes and keys within a dimension for the member keys.
- Use unique conformed values for similar dimensions between the target and source environments when enabling drill through.
- When using a dimensionally model relational model for drill-through, ensure the root business key conforms with the root members of the other data sources.
- Ensure that the business keys and dimension metadata structure are the same between the production and test environments.
- Do not change the business keys in Framework Manager after going into production.
- Resolve the non-unique keys within a dimension in the data source. Tildes are not recommended in the category codes.
- If you have a Transformer cube that is built with non-unique source values, do not use the Clean House feature in Transformer because it will most likely change the category codes. We recommend that you keep a backup copy of your MDL file.

Keys for Levels

A key is a query item that uniquely identifies members in a level. For example, Product Number uniquely identifies a product while City, State, and Country are all needed to uniquely identify a city. The key may or may not be contained in a level. Foreign keys are used to relate the measure dimension to its regular dimensions.

Each level needs an item that is defined as a key.

If a model dimension contains a query item whose data type is `BLOB`, create a query subject that has determinants and then create a model dimension that is based on the model query subject.

Roles

Information about roles for SAP BW metadata appears in a different topic (p. 202).

Roles define what appears in the member tree in the Cognos 8 studios. Use roles to organize and manage metadata and to determine how to present data to your users.

You can also create expressions that refer to roles instead of query items. You must use the `roleValue` function to refer to a particular role. For example, you want to query against a specific role in a hierarchy but the query item playing that role is different at each level of the hierarchy. A single query can span the different query items at each level. You can also use the `roleValue` function when you know the role but not the underlying query item.

You can assign multiple roles to one query item, but the same role cannot be assigned to different query items in the same level.

Default roles are pre-defined for all parent-child hierarchies and for all levels in level-based hierarchies. Most of these roles are not visible in the Cognos 8 studios.

The roles that are reserved by Cognos 8 start with an underscore. The name for a custom role cannot start with an underscore.

Default Roles

The default roles include the following:

- _businessKey

  Represents the key for the level. The level can be defined as unique if the business key of the level is sufficient to identify each set of data for a level.

  The _businessKey role can be assigned to only one attribute in a level.

  The **Root Business Key** property shows the value of the business key for the root member. The root member is an artificial level created for dimensionally modeled relational models. To enable drill-through on conforming dimensions, you must set the **Root Business Key** property.

- _memberCaption

  Presents the caption for a member that will be shown in the Cognos 8 studios.

  The _memberCaption role is necessary to leverage member functions and to enable dragging and dropping levels in the Cognos 8 studios.

  Ensure that the data type is set to string for the item that will be assigned the _memberCaption role.

- _memberDescription

  Returns the description for a member within a dimension.

Custom Roles

By default, attributes are included with no role. You can assign attributes to existing roles or you can create custom roles. Each role that you create must have a unique name.

You can translate the custom roles in the model.

Specify Roles

Roles define what appears in the member tree in the Cognos 8 studios. Use roles to organize and manage metadata and to determine how to present data to your users.

Steps

1. Click the dimension whose roles you want to define.
2. From the **Actions** menu, click **Edit Definition**.
3. Click the **Dimension** tab.
4. In the **Hierarchies** box, click the level you want.
5. In the **Select a level in the hierarchy control to see the query items** box, click a query item.
6. Under **Role**, click the ellipsis (...) button.
7. Do one of the following:
   - To use a role defined by Framework Manager, click the **Default Roles** tab, and select a role.
   - To create a role, click the **Custom Roles** tab, and click **Add**.
8. Click **Close**.
9. Click **OK**.

You can also use the **Dimension Map** tab to define roles. Click **Attributes**, right-click the query item, and click **Edit Roles**.

Create a Measure Dimension

Information about modifying a key figures dimension for SAP BW metadata appears in a different topic (p. 205).

A measure dimension is a collection of facts. You can create a measure dimension for one or more query subjects that have a valid relationship between them.

Model measure dimensions should be composed of only quantitative items. Because, by design, model measure dimensions do not contain keys on which to join, it is not possible to create joins to model measure dimensions. Instead of joining model measure dimensions to regular dimensions, create joins on the underlying query subjects. Then either manually create a scope relationship between them or detect scope if both dimensions are in the same namespace.

Only measures are visible in the model measure dimension. Query items, such as keys, are hidden.

While you can use data source dimensions, they have limited functionality in comparison to query subjects or model dimensions. It is recommended that you discontinue using data source dimensions (both regular and measure). Create new models following the recommendations in "Guidelines for Modeling Metadata" (p. 321) to use query subjects as the relational foundation of the model. Define regular and measure dimensions as model objects based on data source query subjects or model query subjects or both. Guidance on migration for existing users of data source dimensions will be provided in a future release.

You can add value by embedding calculations based on existing business rules, such as Profit Margin.

You can change the order of measures, query items, and calculations.

Constraints

If the measure dimension contains a folder, you can change the order only in the **Project Viewer**.

You cannot define hierarchies or levels for a measure dimension.

Steps

1. Click a namespace where you want to place the measure dimension.
2. From the **Actions** menu, click **Create**, **Measure Dimension**.
3. Click the **Measure Dimension** tab.
4. Drag measures from the **Model Objects** box to the **Measures** box.
5. Perform the actions that you want.

Goal	Action
Embed a calculation	Click **Add**. You can also right-click a measure and click **Add** or **Edit**.
Embed a filter	Click the **Filters** tab.
Test the measure dimension	Click the **Test** tab.
Convert a measure into a query item	Right-click the measure and click **Convert to Query Item**.

Note: If you test the measure dimension by using the **Query Information** tab, Cognos 8 validates the measure dimension. If you test the measure dimension by using the **Test** tab, Cognos 8 executes the measure dimension. The SQL for validate is slightly different than the SQL for execute. To generate definitive SQL for the measure dimension, use the **Test** tab.

6. Click **OK**.

You can also use the **Dimension Map** tab to create a measure dimension. Click the **measure dimension** button .

You may be interested in the following related topics:

- multiple measure dimensions that are related to dimensions at different levels (p. 327)
- embedded calculations (p. 153)
- embedded filters (p. 156)
- testing the measure dimension (p. 128)

Convert a Measure into a Query Item

If you have created a measure dimension and want to join it to regular dimensions, you need to create joins. Joins need keys and keys are query items, not measures. The measure that you want to use as a key must be converted into a query item.

You can also convert a query item into a measure (p. 153).

Steps

1. Double-click the measure dimension that contains the measure.
2. Click the **Measure Dimension** tab.
3. Right-click the measure and click **Convert to Query Item**.
4. Click **OK**.

Scope Relationships

Scope relationships are necessary to define which dimensions and measures are used together for dimensionally modeled relational models.

Scope relationships exist only between measure dimensions and regular dimensions to define the level at which the measures are available for reporting. They are not the same as joins and do not impact the `Where` clause. There are no conditions or criteria set in a scope relationship to govern how a query is formed, it specifies only if a dimension can be queried with a specified fact. The absence of a scope relationship results in an error at runtime.

If you set the scope relationship for the measure dimension, the same settings apply to all measures in the measure dimension. If data is reported at a different level for the measures in the measure dimension, you can set scope on a measure. You can specify the lowest level that the data can be reported on.

When you create a measure dimension, Framework Manager creates a scope relationship between the measure dimension and each existing regular dimension. Framework Manager looks for a join path between the measure dimension and the regular dimensions, starting with the lowest level of detail. If there are many join paths available, the scope relationship that Framework Manager creates may not be the one that you intended. In this case, you must edit the scope relationship.

A scope relationship is automatically generated when you drag a dimension into the dimension map or when you move a query subject into the dimension namespace and convert it to a regular dimension.

Note: Shortcuts to scope relationships are not supported.

Define Scope Relationships

Scope relationships exist only between measure dimensions and regular dimensions to define the level at which the measures are available for reporting.

Steps

1. Click the **Dimension Map** tab.

   Tip: To view scope relationships highlighted with a background color, click the **show scope** button .

2. Click one or more measure dimensions.

3. Click the level of the dimension that you want to set the scope to.

   Tip: If you want Framework Manager to define the scope relationship, select the measure dimension and the regular dimension, and click the **determine scope** button .

4. Click the **set scope** button .

If you want to remove the scope, select the hierarchy or dimension and click the **remove scope** button .

If you select a hierarchy, you can remove the scope from a specific hierarchy without affecting the scope set in other hierarchies of the dimension.

If you select the dimension, all scope from all hierarchies is removed. The scope relationship between the measure dimension and the regular dimension is also removed.

Create a Regular Dimension Based on Existing Objects

You can create a new regular dimension by merging existing objects. These objects can be dimensions, query subjects, or query items.

Steps

1. Select the objects that you want in a dimension.

2. From the **Actions** menu, click **Merge in New Regular Dimension**.

View Related Objects

Information about viewing related objects for SAP BW dimensions appears in a different topic (p. 207).

You can explore a visual representation of the objects that are connected to the query subject or dimension that you select in the **Project Viewer**. The **Context Explorer** shows the objects that the selected object is connected to. You can also select a connected object and see its references.

You can hide an object in the **Context Explorer**. You can also change the layout, fit all objects in the **Context Explorer**, and zoom in and out.

You can also use the **Dimension Map** tab to explore dimensions.

Steps

1. Select one or more objects that you want to explore.
2. From the **Tools** menu, click **Launch Context Explorer**.
3. To see the connected objects, click one or more objects and click the appropriate button.

Goal	Button
View the objects that are related to the selected object.	
View the immediate references for the objects.	
View all references for the objects.	

4. If you want to see details about an object, such as its relationships and query items, right-click the object, click **Diagram Settings**, and then select the details you want.

Test a Dimension

Information about testing SAP BW metadata appears in a different topic (p. 208).

Testing a regular dimension returns the attributes associated with the hierarchy defined as the default.

You can see the results that an object returns by testing it. You can test when creating an object or later on. The objects you can test are dimensions, query subjects, query sets, hierarchies, levels, calculations, and query items.

You can view the data that will appear in a specific report before publishing a package by selecting and testing the objects that will appear in the report. This makes it easier to debug a model and to verify that the model meets the reporting requirements because you do not need to create and publish packages first.

When you test an object, Framework Manager returns sample data. Formatting is not applied to the sample data. If you must test formatting, you must publish the package and view the objects in the Cognos 8 studios.

You may see different results depending on what you test. For example, if you use the expression editor to test a calculation that is embedded in a query subject, Framework Manager tests only the expression, not the item, so the aggregation setting for the query item is not applied to the test.

Testing the entire query subject, which includes the calculation, gives a different result because the aggregation setting is applied. For example, if the aggregation setting is summarize, you can see a smaller number of rows in the test.

If you test a child segment of a segmented model, you may see an error if an object you are testing refers to an object in another child segment and the referenced object is not available to the project you are in. We recommend that you check that the parent model contains all the objects and that this error message does not appear when you test the parent model.

Governor settings may affect the testing results. For more information, see "Set Governors" (p. 310).

You can change existing test settings to customize the results that the test shows. For example, in addition to other settings, you can control the number of rows returned.

Steps When Creating or Modifying the Object

1. Select the object you want to test.
2. From the **Actions** menu, click **Edit Definition**, and click the **Test** or **Query Information** tab.

   The **Test Results** box is initially empty until you run the query.

   Any result sets that contain binary large objects are shown as [blob].
3. To run the query and bring back all the test results, click **Test Sample**.
4. If you want to add a count of the rows, click **Total Rows**.
5. If you want to apply the **Regular Aggregate** property of the query item or the **Aggregate Rules** property of a semi-additive measure that is referenced in the expression, select the **Auto Sum** check box.

   If you clear this check box, a row is returned for each row in the result set of the query.
6. If you want to obtain more information about the query results, click the **Query Information** tab.
7. Click **OK**.

Steps to View the Data That Will Appear in a Specific Report

1. Select the objects that will appear in the report.
2. From the **Tools** menu, click **Test**.

   A message appears, telling you if the test was successful or if problems were found.
3. To view details about any problem that is found, click the **Query Information** tab.

   If you do not see the results of the query in the test window, the data from your data source may exceed the value of the **Runtime Limits** governor. The query stops at the specified limit, but the test result window does not contain any data. **Tip:** Set the **Runtime Limits** governor to zero.

When you test a measure dimension, the SQL uses aggregates not the measures.

Change the Test Settings

You can customize the tests by changing the test settings.

Steps

1. Select the object that you want.
2. From the **Actions** menu, click **Edit Definition** and then click the **Test** tab or the **Query Information** tab.
3. Click **Options** and then click the **Test Settings** tab.
4. Choose the options that you want.

Goal	Action	Persistence
Retrieve all data and show a specified number of rows	Select the **Restrict the maximum number of rows to be returned** check box and type the required number of rows. This setting does not improve performance for retrieving data when testing dimensions, query subjects, and query sets.	This setting applies to all dimensions, query subjects, and query sets in the model. This setting is saved and used in your next session with any model.
Specify the level of detail	Drag the **Level of Information shown in Results Information** slider to the location that represents the amount of detail you require.	This setting is saved and used in your next session with this model.
Temporarily override session parameters	In the **Session Parameters** box, click **Set**. The **Session Parameters** dialog box appears.	The override values are not saved with the model. This setting is for your current session only.
Apply relevant design mode filters	Select the **Apply all relevant design mode filters when testing** check box. This applies all relevant filters whose usage is set to design mode in another dimension, query subject, or query set.	This setting is saved and used in your next session with any model.
Apply a security filter	In the **Security Filters** box, click **Edit**.	This setting is saved and used in your next session with this model.

Goal	Action	Persistence
Change the prompt values	In The **Current Prompt Values** box, click **Prompts**.	The prompt values are not saved with the model.
	The **Model Prompts Manager** dialog box appears, which shows all prompts, and their values, that are in the model.	This setting is for your current session only.

5. Click **OK** two times.

You may be interested in the following related topics:

- working with dimensions (p. 115)
- working with query subjects (p. 87)

Convert a Regular Dimension into a Query Subject

You can convert a regular dimension into a model query subject or a data source query subject.

You can also convert a query subject into a dimension (p. 107).

Constraint

If a dimension has multiple hierarchies, only the default hierarchy is included when you convert the dimension to a query subject.

Steps

1. Click the regular dimension.
2. From the **Actions** menu, click **Convert to Query Subject**.

Multilingual Metadata

For models that are published in multiple languages, you can view and modify model objects in the different languages.

We recommend that you handle multilingual support in the import view for a variety of reasons. You can reduce the number of query items contained in each dimension and query subject. With fewer dimensions, query subjects, and query items, the model is more manageable. You can simplify maintenance by doing all multilingual work in one place instead of in different business views. This ensures consistency because the languages are set up correctly for all modelers to use. This is particularly important for segmented models.

To support multilingual metadata, do the following:

❑ Import metadata from multilingual data sources (p. 133).

- Define the languages the model supports (p. 135).
- Define one or more parameter maps that translate the locale used when the report is run into the language values in the data source (p. 161).
- Use a macro to dynamically substitute language values from the language lookup table using the runLocale session parameter as the key (p. 165).
- Export multilingual properties in translation tables, which translators use to enter the correct text for each language (p. 136).
- Import the table that contains the translated property values (p. 137).
- Publish the metadata in the languages you specify (p. 247).

For information about how to enable multilingual modeling, see "Example - Create a Multilingual Project for Relational Metadata" (p. 137).

Setting Up a Multilingual Reporting Environment

You can create reports that show data in more than one language and use different regional settings. This means that you can create a single report that can be used by report consumers anywhere in the world.

The samples databases provided with Cognos 8 store a selection of text fields, such as names and descriptions, in more than 25 languages to demonstrate a multilingual reporting environment. For information about how data is stored in the samples databases and how the samples databases are set up to use multilingual data, see the *Administration and Security Guide*.

Here is the process for creating a multilingual reporting environment:

- Use multilingual metadata.

  The data source administrator can store multilingual data in either individual tables, rows, or columns.

  For more information about configuring your database for multilingual reporting, see the *Administration and Security Guide*.

- Create a multilingual model.

  Modelers use Framework Manager to add multilingual metadata to the model from any data source type except OLAP. They add multilingual metadata by defining which languages the model supports, translating text strings in the model for things such as object names and descriptions, and defining which languages are exported in each package. If the data source contains multilingual data, modelers can define queries that retrieve data in the default language for the report user.

  For more information, see the Framework Manager *User Guide*.

- Create multilingual maps.

  Administrators and modelers use a Windows utility named Map Manager to import maps and update labels for maps in Report Studio. For map features such as country and city names,

administrators and modelers can define alternative names to provide multilingual versions of text that appears on the map.

For more information, see the Map Manager *Installation and User Guide*.

- Create a multilingual report.

  The report author uses Report Studio to create a report that can be viewed in different languages. For example, you can specify that text, such as the title, appears in German when the report is opened by a German user. You can also add translations for text objects, and create other language-dependent objects.

  For more information, see the Report Studio Professional Authoring *User Guide*.

- Specify the language in which a report is viewed.

  You can use Cognos Connection to do the following:

  - Define multilingual properties, such as a name, screen tip, and description, for each entry in the portal.
  - Specify the default language to be used when a report is run.

    Tip: You can specify the default language on the run options page, in the report properties, or in your preferences.

  - Specify a language, other than the default, to be used when a report is run.

  For more information, see the Cognos Connection *User Guide*.

The data then appears in the language and with the regional settings specified in

- the user's Web browser options
- the run options
- the Cognos Connection preferences

Any text that users or authors add appears in the language in which they typed it.

Modeling with Multilingual Data Sources

To enable a project to work with multiple languages, you must set up data sources to support multiple languages.

Multilingual Relational Data Sources

For relational data sources, you can support multiple languages by using one or more of the following:

- Language-specific database tables

  The data source should contain the same tables for each supported language. For example, if the Product table supports English, French, and German, the data source has tables named Product_en, Product_fr, and Product_de.

- Language-specific columns

A database table should contain the same columns for each supported language. For example, if the Product table supports English, French, and German, the table has columns for ProductName_en, ProductName_fr, and ProductName_de.

- Language-specific rows

  A database table should contain an additional column to identify the language of each row of data, such as a column named LANG.

These solutions can make the multilingual data sources large and difficult to manage.

You can model a single relational query subject to represent all possible data source languages by using parameter maps and session parameters in the query subject definition. For more information, see "Creating Prompts with Query Macros" (p. 165) and "Multilingual Metadata" (p. 131).

Note: Expression syntax is specific to the design language of the model. If you import objects from a model designed in another language, you may have to adjust the expression syntax.

Multilingual SAP BW Data Sources

For SAP BW metadata, you do not need to use parameters to support multilingual reporting. Since SAP BW automatically provides data in the language that matches the logon settings for the current user. If there is no metadata for the current user's language, Framework Manager retrieves data in the default language.

Using a Macro to Model Multilingual Data

You can model multilingual data that is stored in multiple tables, columns, or rows for each supported language. You can use macros with parameter maps and session parameters to create dimensions or query subjects that retrieve data in the preferred language of the person viewing the report.

The location of a parameter in the query subject definition depends on the location of multilingual data in the data source. You must have a parameter map in the macro.

Data source location	Parameter location	Example

A column with a language key in another column	Select list	`Select`
		`PRODUCT_TYPE.PRODUCT_TYPE_CODE,`
		`PRODUCT_TYPE.PRODUCT_LINE_CODE,`
		`PRODUCT_TYPE.PRODUCT_TYPE_`
		`#$Language_lookup`
		`{$runLocale}# as Product_type`
		`from`
		`[gosales].PRODUCT_TYPE`
		`PRODUCT_TYPE`

Data source location	Parameter location	Example

Rows whose language is identified by a special column, such as LANG	Filter	`Select`
		`PRODUCT.PRODUCT_NAME,`
		`PRODUCT_MULTILINGUAL.`
		`PRODUCT_NUMBER`
		`from`
		`[gosales].PRODUCT,`
		`[gosales].PRODUCT_MULTILINGUAL`
		`Where`
		`PRODUCT.PRODUCT_NUMBER =`
		`PRODUCT_MULTILINGUAL.`
		`PRODUCT_NUMBER`
		`and`
		`(PRODUCT_MULTILINGUAL."LANGUAGE" =`
		`#sq($Language_lookup{$runLocale})#`

Add a Language to a Project

You can add a language to a project at any time. For example, you do this if the values for a language were not translated earlier.

When you add a language to a project, Framework Manager generates a new property value for every multilingual property of each object in the project. A multilingual property is any text property that appears in a report, such as **Name**, **Description**, and **Screen Tip**.

The new values that Framework Manager assigns to these text properties are a combination of the original property value preceded by the language code. For example, if a dimension is named Country, and you add the Dutch language, Framework Manager inserts a name whose value is (nl) Country.

Each project contains two types of language definitions:

- design language

  This is the language in which the model was originally created. This value is stored in the model and cannot be changed. It serves as the default language value.

- active language

  This is the language in which model content is currently shown. When you open a model, the active language is set to the language in the model that most closely matches the region and language settings of the computer. You can change this value at any time for your current session only. In future sessions, the model continues to open in the design language.

Steps

1. From the **Project** menu, click **Languages**, **Define Languages**.
2. In the **Available languages** box, select each language you want to add and click the arrow button to move it to the **Project languages** box.

   Tip: To remove a language, select it in the **Project languages** box and click the arrow button to move it to the **Available languages** box.

3. If you want to change the active language, in the **Project languages** box, click a language and click **Set as Active**.
4. Click **OK**.

   At the prompt, accept the changes you made to the project.
5. Click **OK**.
6. If you want to view multilingual property values in the **Properties** pane, click the **Languages** tab.

Export a Translation Table

You can generate and export a translation table to simplify the task of translating model objects. The translation table contains a list of all the text strings defined for multilingual properties, such as **Name**, **Description**, and **Screen Tip**. Translators can then use an external application, such as Microsoft Excel, to type the required information in the table.

You can export a translation table as either a comma-separated value file (.csv) or Unicode text file (.txt). You must export the translation table as a Unicode text file if it either contains a non-Latin language or will be imported by a computer with a language setting that is different from your own computer.

Steps

1. Select the objects you want to export.
2. From the **Project** menu, click **Languages**, **Export Translation File**.
3. In the **Project Languages** box, click the languages you want to export, and click the arrow button to move them into the **Languages to be exported** box.

   You must export the design language of the model that will use the translation table. For example, if the translation table will be used in a model that uses French as the design language, you must export French.

   Framework Manager exports the appropriate locale code for each language you select. If you do not select all the languages to be translated, you must manually enter the language codes in the first row of each new language column in the translation table.
4. In the **Model objects to be exported** box, select whether you want to export all model objects, or export only preselected objects and their children.
5. Enter the location and name of the translation table.
6. Click **OK**.

Import a Translation Table

You can add text property values for each language defined in your model by importing translated data from a file. The imported file must be a translation table that was used by translators to enter the required translated values.

The translation table must contain the design language of the model that will use the translation table. The translation table can contain a subset of the languages defined for the project.

Steps

1. From the **Project** menu, click **Languages**, **Import Translation File**.
2. In the **Project Languages** box, click the languages in the translation table, and click the arrow buttons to move them to the **Translate from** and **Translate into** box.

   You must select the design language for this model.
3. In the **Apply translation to** box, select whether you want to apply the translation to all model objects, or only to preselected objects and their children.
4. Enter the location and name of the translation file.
5. Click **OK**.

Example - Create a Multilingual Project for Relational Metadata

You want to create a model that can be used by English, French, and German report authors. You also want the Cognos 8 studios to automatically show metadata in the language required by the report author.

In the go_data_warehouse sample, you need to do the following:

- Translate the metadata.
- Use macros to create a multilingual project.

  You can modify the dimensions and query subjects to show multilingual content by using the Language_lookup parameter map and the runLocale session parameter.

Steps

1. Open the go_data_warehouse project.
2. Ensure that English, French, and German are supported languages:
   - From the **Project** menu, click **Languages**, **Define Languages**.
   - Ensure that the **Project languages** pane contains English, French, and German.
   - In the **Project Viewer** pane, click a query item and, in the **Properties** pane, click the **Languages** tab.

     For the name, description, and tool tip text, you see one entry for each language.

3. Export all the languages and objects in the project to a comma-separated value file (.csv) named GOSLDW-ML.csv.
   - From the **Project** menu, click **Languages**, **Export Translation File**.
   - In the **Project Languages** box, Ctrl+click **English**, **French**, and **German**, and click the top arrow to move them to the **Languages to be exported** box.
   - In the **Export languages to this file** box, enter the location of GOSLDW-ML.csv.
4. Open the GOSLDW-ML.csv file in Microsoft Excel, and translate the strings.

   Note that each column represents a given language, and the file contains only the text strings that exist in the model.
5. In Framework Manager, import the translated file:
   - From the **Project** menu, click **Languages**, **Import Translation File**.
   - In the **Project Languages** box, move **French** and **German** into the **Translate into** box.
   - In the **Import translation table from this file** box, enter the location of GOSLDW-ML.csv.
6. In the **Project Viewer**, double-click the **Language_lookup** parameter map.

   Note that the keys match the possible values for the runLocale session parameter, which identifies the language of the current user. These keys are mapped to the language values that are defined in the go_data_warehouse database.
7. Ensure that the parameter map contains the following information.

Key	Value
de	DE
en	EN
fr	FR

8. Select an object that contains multilingual columns, such as the Order method dimension, and, from the **Actions** menu, click **Edit Definition**.

   The language identifier that was appended to the query item name is replaced by a parameter map and the runLocale session parameter:

```
Select
 ORDER_METHOD_DIMENSION.ORDER_METHOD_KEY,
 ORDER_METHOD_DIMENSION.ORDER_METHOD_CODE,
 ORDER_METHOD_DIMENSION.ORDER_METHOD_#$Language_lookup{$runLocale}
# AS ORDER_METHOD
from
 [go_data_warehouse].ORDER_METHOD_DIMENSION
```

9. To preview the results, click the **Test** tab and then click **Test Sample**.

10. From the **Project** menu, click **Session Parameters** and change the `runLocale` session parameter value to fr.
11. Test the Order method dimension again to view the results.

Query Items

Information about query items for SAP BW metadata appears in a different topic (p. 213).

A query item is the smallest object in the model that can be placed in a report. It represents a single instance of something, such as the date that a product was introduced.

For relational metadata, you can modify the properties of query items by

- setting **Usage** and **Regular Aggregate** properties to reflect the intended use of the query item (p. 141)
- formatting query items to control how data appears in a report (p. 148)
- identifying a column as a prompt, and controlling how your users see the prompt information (p. 149)

You can also modify the properties for multiple query items at the same time (p. 40).

Because reports can contain different query items from one or more objects in the model, query item properties control many aspects of the final report. When you create a model dimension or model query subject, the query items inherit the properties of the data source query items on which they are based.

The properties for query items or measures include the following.

Query item property	Description
Name	The name of the query item or measure.
Description	A description of the query item or measure.
Last Changed	The date that the query item or measure was last changed.
Screen Tip	A description that can appear in the published package for your users.
Expression	Used to create embedded calculations that provide your users with calculated values that they regularly use. This property is for measures only. **Note:** The **Expression** property is not used by SAP BW.
External Name	The name that appears in the data source.

Query item property	Description
Is Hidden	Whether to hide or show the query item or measure in the published package. Even when **Is Hidden** is set to **True** and the query item or measure is invisible to your users, it is always present in the published package because the query item or measure may be needed by other objects in the model. You do not see the query item or measure in the **Package Publish** wizard. For example, a calculation may make use of a hidden query item.
Usage	The intended use for the data represented by the query item. This property is for query items only.
Format	How information appears in a report.
Currency	Which currency is used. This property cannot be changed in the **Property** pane. Use the **Format** property to change the currency.
Data Type	The data type that was set in the data source. Because this property is set in the data source, it is read-only in Framework Manager.
Precision	The total number of digits. Because this property is set in the data source, it is read-only in Framework Manager.
Scale	How many digits are represented in the scale. For example, you can show numbers in thousands so that 100,000 means 100,000,000. Because this property is set in the data source, it is read-only in Framework Manager.
Size	The size of the query item or measure. Because this property is set in the data source, it is read-only in Framework Manager.
Is Nullable	Whether the query item or measure can contain a null value. Because this property is set in the data source, it is read-only in Framework Manager.

Query item property	Description
Display Type	How the query item is shown. The column value can appear in the Cognos 8 studios as a picture, as a link, or as a value. The default is value. This property is for query items only.
MIME Type	The format that the column value uses. For example, if **Display Type** is set to picture, **MIME Type** could be jpeg. This property is for query items only. **Note:** The **MIME Type** property is not used by SAP BW.
Prompt Info	Prompt behavior.
Regular Aggregate	The type of aggregation that is associated with the query item, measure, or calculation in the published package.
Aggregate Rules	For dimensionally modeled relational metadata, the rules for semi-additive aggregation. For SAP BW metadata, the **Aggregate Rules** property is read-only.
Is Unsortable	Whether the values of this query item can be sorted. This property is for query items that contain large objects such as BLOBs.

You can rename a query item in the **Calculation Definition** dialog box. Renaming the query item updates references to this query item.

You may be interested in the following related topics:

- the **Usage**, **Regular Aggregate**, and **Aggregate Rules** properties (p. 141)
- formatting the query item (p. 148)
- prompts (p. 149)

Modifying How Query Items Are Aggregated

Information about aggregation of SAP BW query items appears in a different topic (p. 216).

You can change how some query items and measures are aggregated in reports. Framework Manager applies aggregate rules when your users create a report that summarizes a query item or measure.

When you import metadata, Framework Manager assigns values to the **Usage** and **Regular Aggregate** properties for query items and measures depending on the type of object that the query item or measure is in. The **Usage** property identifies the intended use for the data represented by the query item (p. 145). The **Regular Aggregate** property identifies the type of aggregation that is applied to the query item or measure (p. 146). Your users can override the values of the **Regular Aggregate** property. For semi-additive measures, you can specify additional aggregate rules by modifying the **Aggregate Rules** property (p. 146).

When modifying the **Regular Aggregate** property, you can select values that are not available through importing, such as average and maximum. You must understand what the data represents to know which aggregate rule is required. For example, if you aggregate a part number, the only aggregate values that apply are count, count distinct, maximum, and minimum.

Rules for Setting Properties for Dimensions

Framework Manager uses the following rules to set the **Usage** and **Regular Aggregate** properties.

Object	Usage property	Regular Aggregate property
Query item in a regular dimension	Attribute	Unsupported
Query item in a measure dimension	Identifier	Count
Measure in a measure dimension	Fact	Automatic if the measure is a calculation Sum if the measure is not a calculation

If the measure is semi-additive, use the **Aggregate Rules** property to define rules for semi-additive aggregation (p. 146).

Rules for Setting Properties for Query Subjects

You can change the **Usage** and **Regular Aggregate** properties for all types of query subjects. The settings for these properties are based on characteristics such as data type and participation in keys and relationships.

For model query subjects, Framework Manager uses the settings of the underlying query subjects. If the source query subject does not use these properties, the rules for data source and stored procedure query subjects are applied.

For data source or stored procedure query subjects, Framework Manager uses the following rules to set the **Usage** and **Regular Aggregate** properties when importing the query subjects.

Object	Usage property	Regular Aggregate property
Query item is part of a key in a determinant	Identifier	Count
Query item participates in a relationship	Identifier	Count
Query item is data type date or time	Identifier	Count
Query item is data type numeric or time interval	Fact	Automatic if the item is a calculation Sum if the item is not a calculation
None of the above applies	Attribute	Unsupported

Rules for Setting Properties for Calculations

Framework Manager uses a number of rules for setting the **Usage** and **Regular Aggregate** properties for calculations.

Rules for Interpreting Calculated Aggregations

The calculated aggregation type is used only for stand-alone calculations. For stand-alone calculations, Framework Manager uses the following rules to interpret the **Regular Aggregate** property:

- Standard aggregation functions (average, count, maximum, minimum, standard deviation, sum, variance) and references to model query subjects are aggregated first. The remaining operations are then applied to the aggregation result.

  For example, to divide debt by credit for each row, the SQL looks like this:

  ```
  Select
  customer, debt, credit, debt/credit as
  percent_debt from x
  ```

  To aggregate for all customers, the SQL looks like this:

  ```
  Select sum(debt), sum(credit), sum(debt)/sum(credit)
  as percent_debt from (Select customer, debt, credit from x)
  ```

- If the query item in the calculation is a fact and the aggregation type for the query item is average, count, maximum, minimum, or sum, the aggregation type of the query item is used.

- If the query item in the calculation has no aggregation type set, the aggregation type minimum is applied in the query. It is not possible in SQL to have an aggregation setting of none.

- Aggregate functions are interpreted as if they are applied to a value in a single row when these functions are used in the detail context. For example, a Report Studio report has the Auto Group and Summarize property set to false.
- Aggregation of a query item is based on the aggregated expression derived from the item definition.

  For example, you want to total this stand-alone calculation:

  ```
  [namespace].[Company].[debt]
  / [namespace].[Company].[credit]
  ```

  The calculation is aggregated with this expression:

  ```
  Total([namespace].[Company].[debt]) / Total([namespace].[Company].[credit])
  ```
- Scalar aggregates, also known as running, ranking, and moving aggregates, are calculated for report granularity unless the `For` clause is explicitly specified.
- Granularity of aggregate functions is set by grouping for determinants or by keys of corresponding levels in the cube.

  For example, `Rank([namespace].[Company].[debt]` is interpreted as `Rank([namespace].[Company].[debt] for Report)`.

Rules to Determine the Automatic Aggregation Type

For stand-alone and embedded calculations, Framework Manager uses one of these rules to determine the aggregation type.

Calculation	Aggregation type
Is based on an expression containing an aggregate function such as average, maximum, minimum, or sum	Calculated
Has an if-then-else operation and the if condition references fact items	Calculated
References a calculation using any type except unsupported	Calculated
Has an aggregation type other than unsupported	Calculated
Is based on an expression that references a model query subject whose usage is set to fact and whose aggregation type is set to average, count, maximum, minimum, or sum but the query item expression does not use an aggregate function	Summarize See below for the rules for summarize.
None of these rules apply	Unsupported

Rules to Determine the Summarize Aggregation Type

For stand-alone and embedded calculations, Framework Manager uses one of these rules to determine the aggregation type.

Calculation	Aggregation type
Is a fact containing of only a reference to a query item whose aggregation type is average, count, maximum, minimum, or sum	Uses the aggregation type of the query item
Numeric or an interval type	Sum
Time, datetime, or date type	Maximum
None of these rules apply	Count

Usage Property

The **Usage** property identifies the intended use for the data represented by each query item. During importing, the **Usage** property is set according to the type of data that the query items represent in the data source.

You need to verify that this property is set correctly. For example, if you import a numeric column that participates in a relationship, the **Usage** property is set to **identifier**. You can change the property.

For relational query items, the value of the **Usage** property depends on the type of database object the query item is based on.

Usage property	Database object	Description
Identifier	key, index, date, datetime	Represents a column that is used to group or summarize the data in a Fact column with which it has a relationship. Also represents an indexed column. Also represents a column that is of the date or time type.
Fact	numeric, timeinterval	Represents a column that contains numeric data that can be grouped or summarized, such as Product Cost.
Attribute	string	Represents a column that is neither an Identifier or Fact, such as Description.

Regular Aggregate Property

The **Regular Aggregate** property identifies the type of aggregation for the query item or calculation when you publish it. Your users can either use this default setting to perform calculations on groups of data, or apply a different type of aggregation.

For example, if the **Regular Aggregate** property value of the Quantity query item is sum, and it is grouped by Product Name in a report, the Quantity column in the report shows the total quantity of each product.

The following aggregation types are supported for relational data sources:

- automatic
- average
- calculated
- count
- count distinct
- maximum
- minimum
- sum

Define Aggregate Rules for Semi-Additive Measures

For measures of dimensionally modeled relational metadata, you can define an aggregate rule for each related dimension. These rules are in addition to the **Regular Aggregate** property and are used to specify how semi-additive measures are to be aggregated with respect to information from that dimension. A semi-additive measure is one that is to be summed for some dimensions, but should not be summed across some other dimensions. For the dimensions over which the measure is not additive, a different aggregation rule must be specified.

Aggregate rules are applied in this order:

- the **Regular Aggregate** property is applied to dimensions that are included in the query but are not referenced in the **Aggregate Rules** dialog box
- the aggregation that is specified in the **Aggregate Rules** dialog box is applied to their specified dimensions, in the order that you specified the rules
- the report-level aggregation that is specified in the query

For example, inventory is recorded in the data warehouse with opening and closing balances at the month grain. If you need to report on inventory for different periods of time, you apply an aggregate that takes the value from the appropriate time slice within the period. For an opening balance, the value is the last balance of the previous month. For a closing balance, the value is the last balance of the current month. In this example, the inventory measure has a **Regular Aggregate** property of total and an aggregate rule for the time dimension with a value of last.

You can have only one aggregate rule for each dimension.

Semi-additive measures are not supported for sparse data. Sparse data is multidimensional data in which many of the combinations of dimension values do not have related fact data.

For SAP BW metadata, the **Aggregate Rules** property value is read-only. The **Semi-Aggregate** property is used instead.

Steps

1. Click the measure you want to work with.
2. In the **Properties** pane, click the ellipsis (...) button in the **Aggregate Rules** property.
3. To add a dimension for this measure, click **Add** and select the dimension.

   You can include a shortcut to a dimension if there is a scope relationship between the measure dimension and the regular dimension.
4. To specify the aggregate operation, click the ellipsis (...) button in the **Aggregation Rules** box.

   The following operations are supported:

   - sum
   - minimum
   - maximum
   - average
   - first
   - last
5. If you want to change the order of the aggregate rules, use the arrow buttons.
6. Click **OK**.

After you set an aggregate rule, the icon for the measure changes.

Semi-Aggregate Property

For relational metadata, the **Semi-Aggregate** property value is set to unsupported and is read-only.

If the value is set to **unsupported** in Framework Manager, the semi-aggregate behavior is ignored in the Cognos 8 studios.

The **Semi-Aggregate** property will not be supported in future releases. Instead, use the **Aggregate Rules** property for semi-additive measures.

Determine Usage and Aggregation Rules

When you use the **Determine Usage** and **Determine Aggregation Rules** commands in Framework Manager, you are resetting the values of the **Usage** and **Regular Aggregate** back to their default values. This overwrites all changes you have made to these properties.

When generating aggregation values, Framework Manager assigns a value that is based on the **Usage** property value and the type of object it is.

Usage property value	Regular Aggregate property value
Identifier	Count
Attribute	Unsupported
Fact	Sum

Steps

1. In the **Project Viewer** pane, click one or more dimensions or query subjects.
2. In the **Properties** pane, click the **Properties** tab.
3. Change the **Usage** property to **unknown**.
4. Change the **Regular Aggregate** property to **unsupported**.
5. From the **Tools** menu, click **Determine Usage**.
6. From the **Tools** menu, click **Determine Aggregation Rules**.

Format Query Items

Information about formatting query items based on SAP BW metadata appears in a different topic (p. 219).

You can specify how query item values appear in reports. Use the **Format** property to choose a format type, such as text, date, and currency. Each format type contains properties that further specify how the data appears.

For example, you can assign the **Currency** format type to a numeric query item, and then use the **No. of Decimal Places** property in the **Data Format** dialog box to specify how many decimal places appear in reports.

Some characters are language-sensitive and appear properly only when your locale supports the applicable font. For example, for Japanese currency symbols to appear correctly, your locale must be set to Japanese.

If Framework Manager does not show the currency you require, you must ensure that you install the appropriate language packs to support the currency symbols. For example, to have the Indian currency symbol (rupee) appear, you must run an operating system or install a language pack that can show this symbol. The Japanese operating system or Japanese language is one that can show the Indian currency symbol.

You can define properties for several query items at the same time. However, if the query items have different format types, all properties that were previously specified are overridden and the default values from the data source are used. If the original format types of the selected query items are the same, all the properties for the selected query items are set identically.

For example, to use the same decimal separator for two query items and to keep the number of decimals different, each query item must be changed individually. If both are selected and changed at the same time, all properties including the number of decimals are set identically for both query items.

Steps

1. In the **Project Viewer** pane, click the query item you want to format.
2. In the **Properties** tab of the **Properties** pane, click the **Format** property.
3. Select the appropriate **Format type**.
4. In the **Properties** box, select or type the appropriate property value.
5. Click **OK**.

Define a Prompt Control

Information about prompt controls for SAP BW metadata appears in a different topic (p. 220).

Prompts help your users quickly find the information they need in a report. Prompts are generally defined in reports. However, you can change the definition of the dimension or query subject in the model so that a prompt appears automatically when your users create filters. This is useful for query items, such as ProductTypeCode, whose values are not shown in a report but are useful for filtering data.

In general, it is better to define type-in prompts in the reports to make use of the additional prompt features. However, your users cannot modify some variables. For these variables, you can use Framework Manager to define type-in prompts.

The syntax for using a prompt as a value is
`?<PromptName>?`

You can use prompts in

- parameter maps
- session parameters
- stored procedure arguments
- expressions, including filters, calculations, and relationships

For example, a stored procedure returns all rows with a specific product number. Instead of using the product number as the argument for the stored procedure, you can use a prompt, such as `?Product_Number?`.

For examples of prompts, see "Creating Prompts with Query Macros" (p. 165) and "Example - Use Prompts with a Stored Procedure" (p. 94).

Steps

1. Click the query item.

2. In the **Properties** pane, click the **Properties** tab.
3. Click the plus sign (**+**) next to the **Prompt Info** property.

   This is a compound query item property.
4. Modify the following properties to reflect the behavior you require.

Goal	Property
Set the type of prompt control that is generated when the report is run.	Prompt Type
Set the generated prompt as part of a series of generated cascading prompts.	Cascade On Item Reference
Specify that a prompt defined for one query item will use a different query item, and its rules, in the prompt. The query item specified in the **Display Item Reference** property will display a list of values in the prompt. Each value in the list will be associated with a value in the query item specified in the **Use Item Reference** property.	Display Item Reference
Specify that a prompt defined for one query item will use a different query item, and its rules, in the prompt. The query item specified in the **Use Item Reference** property will be used in the prompt.	Use Item Reference
Set an alternative query item that is used by any Report Studio or Query Studio filter.	Filter Item Reference

Prompt Type Property

The **Prompt Type** property sets the type of prompt control that is generated when the report is run, such as an edit box or a pull-down list. Use this property to have the prompt show one query item but use a different one. For example, the prompt can show Employee Name but use Employee Number.

The default value for this property is **Server Determined**.

Value	Prompt Control
Server Determined	The type of prompt control is based on information in the server, such as the data type of the query item.

Value	Prompt Control
Edit Box	A simple text box. If the data type of the column is date or dateTime, this value generates a date or date-time control as well as the text box.
Select Date	A date control with a calendar interface.
Select Date/Time	A date-time control with a calendar interface. For SAP BW metadata, this value is not relevant.
Select Interval	A date-time interval control. For SAP BW metadata, this value is not relevant.
Select Time	A time control that filters data based on the selected time period. For example, if you define a Select Time prompt for Order Time, the user can use the time control to show all orders placed after 1:00, or all the orders placed between 10:00 and 11:00. If you are referring to a time member, you must use the exact values only. If you are using a range, the end points of the range must correspond to values in the data source.
Select Value	A drop-down list.
Select with Search	A list control so that users can search for values. For SAP BW metadata, this value is not relevant.
Select with Tree	A tree prompt control for prompts that are based on a hierarchy node.

Cascade On Item Reference Property

The **Cascade On Item Reference** property indicates that the generated prompt is part of a series of generated cascading prompts. The query item that you reference in this property is the parent item in the cascade. The system prompts the user for the cascade item before prompting them for the current query item.

For example, if you want to prompt for Product Line and then Product within the selected line, set the **Cascade On Item Reference** property of the Product query item to Product Line.

Display Item Reference and Use Item Reference Properties

These properties specify that a prompt defined for one query item will use a different query item, and its rules, in the prompt.

The query item specified in the **Display Item Reference** property will display a list of values in the prompt. Each value in the list will be associated with a value in the query item specified in the **Use Item Reference** property.

The query item specified in the **Use Item Reference** property will be used in the prompt.

For example, you want the prompt to display Country Name while using Country Code to retrieve data. Set the **Display Item Reference** property to Country Name and set the **Use Item Reference** property to Country Code.

These properties are used only for data driven prompt controls whose **Prompt Type** property is set to either **Select Value** or **Select with Search**.

Filter Item Reference Property

The **Filter Item Reference** property identifies an alternative query item that is used by any Report Studio or Query Studio filter. This property is used only when Report Studio or Query Studio generates a filter. This property helps create more efficient queries because it is faster to filter on an indexed numeric column than a non-indexed string column.

For example, a report author wants to create a filter for the Country query item. You set the **Filter Item Reference** property to use Country Code for any filter that uses the Country query item.

Another example is a report author wants to create a filter for the Country Code query item that appears in the Orders table. You want that filter to use the Country Code in the Country table because there are fewer rows to read in the Country table.

Testing a Prompt

When you test a model object that references a prompt, Framework Manager asks you to enter the prompt value. Framework Manager uses this value for either the duration of the session, or until you clear the prompt value.

You can change the session value of prompt values through the **Options** dialog box. This dialog box is available when you modify a dimension or query subject, or define a calculation, filter, query set, or complex relationship. You can change the prompt value at the time that you are testing the expression that references that value.

If you select the **Always prompt for values when testing** check box in the **Prompt** dialog box, Framework Manager prompts you for a value every time you test the object. When updating the object or performing a count, Framework Manager uses the existing prompt value, if one exists.

A prompt on a query item in a model query subject is associated only with that query item. A prompt on a query item in a data source query subject is associated with the entire query subject and therefore, the prompt appears when you test any query item in the query subject.

Convert a Query Item into a Measure

You can convert a query item in a measure dimension back into a measure.

You can also convert a measure to a query item (p. 126).

Steps

1. Double-click the measure dimension that contains the query item.
2. Click the **Measure Dimension** tab.
3. Right-click the query item and click **Convert to Measure**.
4. Click **OK**.

Adding Business Rules

Information about business rules for SAP BW metadata appears in a different topic (p. 231).

You can add business rules to the dimensions and query subjects in your model to refine the data retrieved and ensure that the correct information is available for your users.

Creating business rules and storing them in the model instead of in reports has many advantages. You save time because you and your users do not have to re-create the business rules every time they are needed. The business rules ensure consistency because your users all use the same definitions. For example, Low Margin means the same thing throughout the organization. They are easy to update because you maintain the business rules centrally so that all reports are updated automatically as the rules evolve. For example, if the definition for Low Margin changes, all reports that use the Low Margin calculation are updated automatically. The business rules enhance security. For example, you can use filters to limit the data that your users can see.

You can

- add calculations so that your users can include calculated data in their reports (p. 153)
- create and apply filters so that you limit the data that a query subject retrieves (p. 156)
- add prompts that will automatically appear whenever a dimension or query subject is used in a report; report consumers are then prompted to filter data (p. 149)
- use session parameters (p. 163) and parameter maps (p. 161) in macros (p. 165) to dynamically resolve expressions
- create a security filter to control the data that is shown to your users when they set up their reports (p. 253)

Create a Calculation

Information about calculations for SAP BW metadata appears in a different topic (p. 231).

You can create calculations to provide your users with calculated values that they regularly use. Calculations can use query items, parameters, variables, calculated members, expressions, and expression components, such as functions.

Punctuation characters, such as the question mark (?), must be in 7-bit ASCII character code. If you type a punctuation character from a multi-byte enabled keyboard, ensure that you type the 7-bit ASCII representation of the character. For example, type Alt+063 for the question mark.

Avoid using characters that are used for expression operators in the name of the calculation. Syntax errors may occur when the expression is evaluated. For example, a calculation named Margin * 10 causes errors when used in an expression such as [Margin * 10]< 20.

At query time, Framework Manager returns a null value for any calculation that contains a divisor whose value is zero. Framework Manager cannot detect zero-division errors in functions such as average and mod, because the division operator is not explicit.

Framework Manager supports stand-alone calculations and embedded calculations.

- Use a stand-alone calculation when you want to reuse the expression.

  You can apply a stand-alone calculation to one or more dimensions or query subjects to provide calculated data to a report, or include it in a package to make it available to your users. By moving a stand-alone calculation or a shortcut to it into a folder, you can better organize the model objects.

  You cannot use stand-alone calculations in Analysis Studio. Instead, use an embedded calculation.

- Use an embedded calculation when you want to use a calculation with only one dimension or query subject.

  You can create an embedded calculation when modifying a relational data source query subject, model query subject, or dimension.

  If you start with an embedded calculation, you can later convert it into a stand-alone expression that you can apply to other dimensions or query subjects. **Tip:** Right-click the calculation expression in the **Calculations** tab and click **Convert to Stand-Alone Calculation**.

  When you embed a calculation, the data source query subject must have a relationship to any query subject referenced by the expression. This relationship is necessary even if the expression references a model query subject based on the same table as the data source query subject in which you are embedding the expression.

  To create a calculation on an unrelated query subject, do one of the following:

  - Ensure that there is a join path between the new query subject and the one that contains the calculation.
  - Base the embedded calculation on a query item that is based on the data source query subject you want.
  - Convert the calculation to a stand-alone calculation, so that it is not part of the query subject.

- Create a stand-alone calculation that references the embedded object.

Steps

1. Do one of the following:
   - If you want to create a stand-alone calculation, click the namespace or folder and, from the **Actions** menu, click **Create**, **Calculation**.
   - If you want to create an embedded calculation, double-click the dimension or query subject that will contain the calculation, click the **Calculations** tab, and then click **Add**.
2. In the **Name** box, type a name for the calculation.
3. Define the expression.

Goal	Action
Add items	On the **Model** tab, click a query item, filter, or calculation and click the arrow.
Add functions	On the **Functions** tab, choose a component and click the arrow.
Add parameters	On the **Parameters** tab, click a parameter and click the arrow.
Retrieve all data and show a specified number of rows	Click the options button, select the **Restrict the maximum number of rows to be returned** check box, and type the required number of rows to be returned. This setting does not improve performance for retrieving data when testing dimensions, query subjects, and query sets.
Override session parameters	Click the options button, click **Set**, enter a value in the **Override Value** field, and click **OK**.
Override prompt values	Click the options button, and then click **Prompts**. The **Model Prompts Manager** dialog box appears, which shows all prompts, and their values, that are in the model.

4. To test the calculation, click the **test** button .

   You can test only calculations that contain query items. If a calculation contains a function, for example _add_days, the test sample button is not available.

5. Click **OK**.

6. Modify the **Data Type** property to identify the type of data the calculation returns.

   The Cognos 8 studio uses this information to format the data that the calculation returns.

You may be interested in the following related topics:

- functions (p. 387)
- overriding session parameters (p. 163)
- testing (p. 128)

Create a Filter

Information about filters for SAP BW metadata appears in a different topic (p. 233).

A filter is an expression that specifies the conditions that rows or instances must meet to be retrieved for the dimension, query subject, calculation, or report to which the filter is applied. A filter returns a boolean value so that you can limit the rows returned by a dimension or query subject.

For example, you can use the `in_range` function to create a filter that retrieves data for products introduced in a specific time frame. The syntax for this example looks like this:

```
[gosales_goretailers].[Products].[Introduction
date]
in_range {Feb 14, 1999 : July 14, 2007}
```

Note: When using a date or time function, you must use a 24-hour clock. Framework Manager does not support "a.m." or "p.m." in expressions. For example, use 20:00 to signify 8 p.m.

You can restrict the data represented by dimensions or query subjects in a project by creating a security filter. The security filter controls the data that your users can see when they set up their reports.

You can also apply governors to restrict the data that the queries in a package retrieve.

Framework Manager supports stand-alone filters and embedded filters.

- Use a stand-alone filter when you want to reuse the expression.

  You can add a stand-alone filter to one or more dimensions or query subjects to limit the data that the query retrieves when the filtered dimension or query subject is used in a report, or you can include it in a package to make it available to your users. By moving a stand-alone filter or a shortcut to it into a folder, you can better organize the model objects.

- Use an embedded filter when you want to use a filter with only one dimension or query subject.

  You can create an embedded filter when modifying a dimension, relational data source query subject, or model query subject.

If you start with an embedded filter, you can later convert it into a stand-alone expression that you can apply to other dimensions or query subjects. **Tip:** Right-click the filter expression in the **Filters** tab and click **Convert to Stand-alone Filter**.

When you embed a filter, the data source query subject must have a relationship to any query subject referenced by the expression. This relationship is necessary even if the expression references a model query subject based on the same table as the data source query subject in which you are embedding the expression.

To create a filter on an unrelated query subject, do one of the following:

- Ensure that there is a join path between the new query subject and the one that contains the filter.
- Base the embedded filter on a query item that is based on the data source query subject you want.
- Convert the calculation to a stand-alone filter, so that it is not part of the query subject.
- Create a stand-alone filter that references the embedded object.

Steps

1. Do one of the following:
   - If you want to create a stand-alone filter, click the namespace or folder and, from the **Actions** menu, click **Create**, **Filter**.
   - If you want to create an embedded filter, double-click the dimension or query subject that will contain the filter, click the **Filters** tab, and then click **Add**.
2. In the **Name** box, type a name for the filter.
3. Define the expression.

Goal	Action
Add query items and filters	On the **Model** tab, drag the objects you want to the **Expression Definition** box.
Add functions	On the **Functions** tab, drag the functions to the **Expression Definition** box.
Add parameters	On the **Parameters** tab, drag the parameters to the **Expression Definition** box.

Goal	Action
Retrieve all data and show a specified number of rows	Click the options button, select the **Restrict the maximum number of rows to be returned** check box, and type the required number of rows to be returned. This setting does not improve performance for retrieving data when testing dimensions, query subjects, and query sets.
Override session parameters	Click the options button, click **Set**, enter a value in the **Override Value** field, and click **OK**.
Override prompt values	Click the options button, and then click **Prompts**. The **Model Prompts Manager** dialog box appears, which shows all prompts, and their values, that are in the model.

4. To test the filter, click the **test** button .
5. Click **OK**.

You can also apply governors to restrict the data that the queries in a package retrieve (p. 310).

You may be interested in the following related topics:

- security filters (p. 253)
- functions (p. 387)
- parameters (p. 161)
- session parameters (p. 163)
- testing (p. 103)

Apply a Filter

Information about filters for SAP BW metadata appears in a different topic (p. 235).

To apply a filter, you must modify the dimension, data source query subject, or model query subject. The query subject must either contain the query items that the filter references, or have a relationship path to the query subjects that contain the query items.

You can embed a stand-alone filter in dimensions or query subjects, but if you want a different usage for each embedded filter, you must create different versions of the stand-alone filter. Otherwise,

your users could be required to fill in a prompt that you thought was optional if there is any instance where the usage is set to mandatory.

For example, in query subject A, you embed a stand-alone filter and define it as optional. In query subject B, you define it as mandatory. When your users create a report that uses both query subjects, they are required to choose values in both filters, even the one defined as optional. All instances of the filter are considered to be mandatory when used in the same query. The solution is to create different versions of the filter, each with its own name.

Steps

1. Create a filter.
2. Select the filter and, from the **Actions** menu, click **Edit Definition**.
3. Click the **Filters** tab, and drag the filter you created to the **Filters** box.
4. Select a usage value for the filter.

Usage Value	Description
Always	Use this usage value to ensure specified data is filtered out of all reports. For example, your company may have obsolete information that it stores but does not want to report on. **Always** is the default usage value.
Design Mode Only	Retrieves a small subset of the data for the sample report. Use this usage value when you do not need to see all the data, for example when testing a query subject. Your users may need the design mode filter in Query Studio when they want to focus on designing the layout and format of a report and not retrieve all the data as they work. To access the design mode filter in Query Studio, run the report with limited data.
Optional	Specifies that the filter is optional. The user is prompted to filter data and can leave the prompt blank. If the prompt is blank, Framework Manager ignores the filter and retrieves all data for the dimension or query subject. The `? ?` syntax is required for optional prompts. Use this usage value if your users want to control when the filter is applied. For example, you want to see on one country sometimes and see the data for all countries at other times. An optional filter for country looks like this: `([GeoNamespace].[Countries].[CountryName] = ?WhichCountry?)`

5. If you want to view the SQL, click the **Query Information** tab.
6. Click **OK**.

Example - Show the Currency Name for Each Country

You want to create a query that shows the currency name for a specific country. To do this, you create a filter that returns data for a specific country code, and apply the filter to a model query subject that retrieves the currency name for each country.

Note: The following example uses a relational data source.

Steps

1. Open the go_sales sample model. It is located in *c8_location*/webcontent/samples/Models/go_sales/go_sales.cpf
2. Create a filter to limit the retrieval of data to only those country codes in the conversion rate table whose value is 2:
   - Click the Filters folder and, from the **Actions** menu, click **Create, Filter**, and name the new filter **ConversionRateCountryCode**.
   - Click the **Model** tab.
   - In the **Available Components** box, open the Database view folder and then open the GoSales folder.
   - Add the Country Code query item from Conversion Rate query subject to the **Expression definition** box, and type **='2'** at the end of the expression.
   - Click **OK**.
3. Create a model query subject named **ISO Code**.
   - In the **Available Model Objects** box, open the Database view folder.
   - Add Country query item and the ISO 3-letter code query item from the Country query subject to the **Query Items and Calculations** box.
4. Apply the ConversionRateCountryCode filter:
   - Click the **Filters** tab.
   - Open the Filters folder and drag ConversionRateCountryCode to the **Filters** box.
5. Click the **Query Information** tab.

   The generated SQL contains the filter even though it does not affect the result set.
6. Change the usage of the ConversionRateCountryCode filter to **Optional**:
   - Click the **Filters** tab.

- Click the ellipsis (...) button under **Usage** for the ConversionRateCountryCode filter, and click **Optional**.

7. If you want to view the SQL, click the **Query Information** tab.
8. Click **OK**.

Create a Parameter Map

Information about parameter maps for SAP BW metadata appears in a different topic (p. 237).

Use parameters to create conditional query subjects that allow for substitutions when the report is run. Parameter maps are objects that store key-value pairs. Parameter maps are similar to data source look-up tables. Each parameter map has two columns, one for the key and one for the value that the key represents. You can manually enter the keys and values, import them from a file, or base them on existing query items in the model.

All parameter map keys must be unique so that Framework Manager can consistently retrieve the correct value. Do not place quotation marks around a parameter value. You can use quotation marks in the expression in which you use the parameter.

The value of a parameter can be another parameter. However, you must enclose the entire value in number signs (#). The limit when nesting parameters as values is five levels.

When you use a parameter map as an argument to a function, you must use a percentage sign (%) instead of a dollar sign ($).

We recommend that you assign an alias to a query item that uses a parameter map as part of its name and to add the multilingual names to the object in the **Language** tab (**Properties** pane).

Constraint

We recommend that you do not base a parameter map on a query item or table with a large result set, such as 50,000 rows. Each time you use the parameter map in an expression or in SQL, Framework Manager executes this large query. Performance is then slowed. Parameter maps are recommended for smaller lookup tables.

Steps to Manually Create a Parameter Map

1. Click the **Parameter Maps** folder and, from the **Actions** menu, click **Create, Parameter Map**.
2. In the **Name** box, type a name for the new parameter map.
3. Click **Manually enter the parameter keys, and/or import them from a file** and click **Next**.
4. Do one of the following:
   - To manually enter values, click **New Key**, type a key, and press Tab to enter a value for that key.
   - To import keys and values, click **Import File** and identify the location of the appropriate .csv file. You can import from a .txt file only if the values are separated by tabs rather than commas.

Note: If you are going to use a parameter in a data source query subject, the value must use English-specific punctuation. This means that you must use a period (.) to represent a decimal and a comma (,) to separate lists of values.

5. Modify existing parameters as required.

Goal	Action
Assign a default value	In the **Default Value** box, type a value. If the key is used in an expression that is not mapped, the default value is used.
Remove a parameter	Select a row and click **Delete**.
Modify a parameter	Select the row you want to modify, click the **Edit** button, and type a value.
Clear all keys and values	Click **Clear Map**.

6. Click **Finish**.

Steps to Base a Parameter Map on Existing Query Items

1. Click the **Parameter Maps** folder and, from the **Actions** menu, click **Create**, **Parameter Map**.
2. In the **Name** box, type a name for the new parameter map.
3. Click **Base the parameter map on existing Query Items** and click **Next**.
4. Click the query item to use as the key, and then click the query item to use as the value.

   Both query items must be from the same query subject.
5. Click **Next**.

   You can assign a default value. If the key used in an expression is not mapped, the default value is used.
6. Click **Finish**.

You may be interested in the following related topics:

- using parameters (p. 164)
- using a parameter map to specify the language value (p. 162)

Example - Specifying a Language Value for Relational Metadata

An international company stores its product information in English and French. With the use of a parameter map and macros, employees can retrieve data that matches the information they require.

Create a Language_lookup parameter map that contains the following:

Key	Value
en	EN
fr	FR

When you examine the SQL for the Product Line query subject, you see the following:

```
Select
 PRODUCT_LINE.PRODUCT_LINE_CODE,
 #'PRODUCT_LINE.PRODUCT_LINE_' + $Language_lookup{$runLocale}#
as Product_Line
from
 [gosales].PRODUCT_LINE PRODUCT_LINE
```

The runLocale macro returns a locale setting that is used by the Language_lookup macro to specify a language value.

Create a Session Parameter

Information about session parameters for SAP BW metadata appears in a different topic (p. 238).

A session parameter is a variable that Framework Manager associates with a session. For example, user ID and preferred language are both session parameters. Because session parameters are key and value pairs, you can think of each session parameter as an entry in a parameter map named Session Parameters. You use a session parameter in the same way that you use a parameter map entry, although the syntax for session parameters is slightly different.

There are two types of session parameters: environment and model.

Environment session parameters are predefined and stored in Content Manager. By default, the following session parameters appear in Framework Manager:

- runLocale
- account.defaultName
- account.personalInfo.userName

If you log on anonymously, you will see only runLocale and account.defaultName(Anonymous).

If your authentication source supports other parameters and you entered information about them in the authentication source, you see other session parameters, such as account.personalInfo.email.

You can define additional parameters by using model session parameters. Model session parameters are stored in a parameter map named **_env**. They are set in the project and can be published with a package.

Model session parameters must have their values set within the scope of objects in the Framework Manager model. The scope can include the use of existing environment session parameters, as well as static values.

Each session parameter must have a name and a default value. You can define an override value to test the results that value returns. The override value is valid only when you have the model open, and is not saved when you save the model. If no override value exists, Framework Manager uses the default value when it executes a query that contains a session parameter.

The rules governing the use of parameters include the following:

- All possible return values must have the same data type.
- Only one value can be defined.

Steps

1. From the **Project** menu, click **Session Parameters**.
2. Click **New Key** and type a session parameter key and value.
3. Choose how to handle the override value.
   - To avoid having to set the override value every time you edit the project, set the session parameter as a value.
   - To avoid having to remove the project setting each time before you publish it, set the session parameter as a session override.
4. Modify existing parameters as required.

Goal	Action
Change the parameter value	Click the row that contains the value you want to change, click **Edit**, and type a value.
Assign a default value	In the **Default Value** box, type a value. Framework Manager uses the default value if a key has an invalid value.
Remove a parameter	Click a row and click the **Delete** button. You cannot delete an environment session parameter.
Clear an override value	Click a row and click **Clear Override**.

5. Click **Finish**.

Using Parameters with Relational Data Source Query Subjects

Model objects do not reflect changes to the data source objects on which they are based. Therefore, when you add a parameter to a data source query subject, consider whether you want to create a model object that references the parameter. If so, you must assign an alias to the parameterized

object in the data source query subject. This ensures that model query subjects, filters, or calculations that reference the object return the correct results when the parameter value changes.

For example, the following SQL defines a data source query subject that contains a session parameter named runLocale. The runLocale parameter value specifies which column the query retrieves. The alias behaves like a shortcut so that when a model object references CountryNameAlias, Framework Manager retrieves the value to which the alias is assigned.

```
Select
 #$ColumnMap{$runLocale}# as
CountryNameAlias
From
 [GoSales].Country
```

Creating Prompts with Query Macros

Macros are fragments of code that you can insert anywhere in the `Select` statement that defines a query subject. You can include references to session parameters, parameter maps, and parameter map entries. Parameter values are set when you run the query.

For example, you can use the language session parameter to show only the data that matches the language setting for the current user.

Macros can be used in these different ways:

- They can be inserted in the SQL.

  An example is `Select * from Country where Country.Name = #$myMap{$runLocale}#`

- They can supply an argument to a stored procedure query subject.

  If a value is not hard-coded for the argument, the stored procedure query subject can be used to return different data.

- They can be inserted in expressions, such as calculations and filters.

  An example is a filter `[gosales].[Sales staff].[Staff name] = #$UserLookUpMap {$UserId}#`

- They can be used to dynamically complete the properties of a data source query subject.

  This enables different users to supply different connection information and thus access different data sources. The properties that can contain macros are: **Content Manager Datasource**, **Catalog**, **Cube**, and **Schema**.

  An example using the **Content Manager Datasource** property is `#$DataSourceMap{$UserId}#`

- They can be used as a parameter wizard.

  Parameters can reference other parameters. An example is `Map1, Key = en-us, Value = #$myMap{$UserId}#`

- They can be used in the **Session Parameter** dialog box.

  An example is `MySessionParameter, value = #$myMap{$UserGroup}#`

You can replace the following query subject elements with a parameter.

Element	Example
Query items identified in the `Select` list	`#'Product_name_'+ $languageCode#`
Tables identified in the `From` clause	`Product_#$language#`
`Where` clause	`Product_lang = #sq($languageCode)#`
Name of the data source, schema, or source property	`#$data_source#.#$schema#.Products`

Syntax

Use the following syntax to reference session parameter and parameter values.

Object	Syntax	Example
Session key	`$session_key`	`#$my_account#`
Parameter map key	`$map{<key>}`	`#$map_one{'abc'}#`
Parameter map entry whose key is defined by a session parameter	`$map{$session_key}`	`#$map_one{$my_account}#`

You can add the following elements to further define the macro expression.

Symbol	Purpose
Single quotation marks '	Delineates a literal string that has a single quotation mark as part of the string. If the single quotation mark appears in a string, such as a query item, the string must be enclosed in a single quotation mark on both sides of the string and the single quotation mark must be doubled. For example, ab'c is written as `'ab''c'` If the single quotation mark appears in a macro, the string must be enclosed in square brackets. For example, ab'c is written as `[ab'c]` If the single quotation mark appears in a prompt, there is no need to enclose the string. To escape a single quotation mark in an expression, use `'`

Symbol	Purpose
Square brackets [ ]	Encloses model objects, such as a namespace or query subject and macro names that contain restricted characters, such as a number sign, hyphen, or space.
Curly brackets, also known as braces { }	Calls a function that is unknown to the parser, such as `dateadd` in DB2, and whose first argument is a keyword. Example: `dateadd ({month},2,<date expression>)`
+ operator	Concatenates two strings, such as `'abc' + 'xyz'`

Single quote function (sq)	Surrounds the result of a string expression with single quotation marks. Existing single quotation marks are double-escaped so that they do not interfere with the quotation structure. You can use this function to build clauses to test against literal parameter-driven values.
	Here is an example:
	`#sq($my_sp)#`
	If a session parameter (my_sp) has the value ab'cc, the result is
	`'ab"cc'`

Double quote function (dq)	Surrounds the result of a string expression with double quotation marks. You can use this function to refer to table and column names with non-identifier characters, such as a blank space or a percent sign (%).
	Here is an example:
	`#dq ('Column' + $runLocale)#`
	If runLocale=en-us, the result is
	`"Column en-us"`

Square bracket function (sb)	Inserts a set of square brackets into the argument to build object references in a model query and model expressions, such as filters and calculations.
	Here is an example:
	`#sb ('my item in ' + $runLocale)#`
	If runLocale=en-us, the result is
	`[my item in en-us]`

For information about functions, see "Using the Expression Editor" (p. 387).

Create a Macro

Macros are fragments of code that you can insert anywhere in the `Select` statement that defines a query subject. You can include references to session parameters, parameter maps, and parameter map entries. Parameter values are set when you run the query.

Constraints

When you reference a parameter, you must do the following:

- use a number sign (#) at the beginning and end of each set of one or more parameters.

  Everything between the number signs is treated as a macro expression, which is processed at runtime. Framework Manager removes anything that is outside the number signs.

- precede each parameter map entry with a dollar sign ($)

- use a name that starts with an alpha character (a..z, A..Z)

Do not insert macros between existing quotation marks or square brackets because Framework Manager does not execute anything within these elements.

Steps

1. Select the data source query subject you want to modify.
2. From the **Actions** menu, click **Edit Definition**.
3. On the **SQL** tab, click **Insert Macro** to start the Macro Editor.
4. In the **Available components** box, click the parameter maps, session parameters, or functions you want to use, and drag them to the **Macro definition** box.

   Ensure that you type the macro expression between the number signs. If you enter text before or after the number signs, when you click OK, Framework Manager deletes this text.
5. Insert single or double quotation mark functions.

   Tip: Click the arrow next to these buttons for a menu of choices for placing the quotation marks .
6. If you want to edit a parameter map or session parameter, click it in the **Macro definition** box.

   The **Parameter Map** or **Session Parameters** dialog box appears. You can set override values for session parameters, add new items, or change values.
7. Check the macro in the **Information** box.

   If a macro is incorrect, an error message appears.

   Tip: To clear a macro, click the **clear all** button .
8. Click **OK**.

Mandatory and Optional Prompts

You can create mandatory and optional prompts in Framework Manager models by using query macros. You can use two prompt macro functions, `prompt` and `promptmany`, to create single value prompts and multiple value prompts. You can insert a prompt macro anywhere in the SQL statement that defines the query subject.

If you want to use a prompt macro in an expression such as a calculation, you must specify the data type when using an overloaded operator, such as a plus sign (+). You can use the plus sign to concatenate two items and to add two items.

If you want to define a filter on a dimension level and have the filter use the `prompt` or `promptmany` macro, you must provide the data type as `memberuniquename` and a default value. Here is an example:

```
members( [MS_gosales].[New Dimension].[PRODUCTLINE].[PRODUCTLINE]
)  in ( set( #promptmany('what', 'memberuniquename', '[MS_gosales].[PROD1].
[PRODUCTLINE].[PRODUCTLINE]->[all].[1]'
)# ) )
```

Here is an example of a mandatory prompt:

```
select
 COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
 COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
 COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
 COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
 gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
where COUNTRY_MULTILINGUAL.COUNTRY = #prompt('CountryName')#
```

The `prompt` and `promptmany` functions have the following parameters. All argument values must be specified as strings.

Parameter	Description
Name (mandatory)	Name of the prompt. Can also refer to the name of a parameter on a user-created prompt page, in which case the user-created prompt page appears when the report is run instead of the default prompt page that the macro would generate.
Datatype (optional)	Prompt value data type. The default value is string. Prompt values are validated. In the case of strings, the provided value is enclosed in single quotation marks and embedded single quotation marks are doubled.

Parameter	Description
	Values include: • boolean • date • datetime • decimal • double • float • int • integer • interval • long • memberuniquename -- Memberuniquename is not an actual data type. Use it in cases when the data type of the parameter is a member unique name (MUN). For example, use the syntax #prompt('Parameter', 'memberuniquename')#. • numeric • real • short • string • time • timeinterval • timestamp • token -- Token is not an actual data type. It is a way to pass SQL. A token does not pass values. • xsddate • xsddatetime • xsddecimal • xsddouble • xsdduration • xsdfloat • xsdint • xsdlong • xsdshort • xsdstring • xsdtime
DefaultText (optional)	Text to be used by default. If a value is specified, the prompt is optional. If you use a space and no values are provided in the **Prompt Value** dialog box, a `Where` clause is usually not generated. If you use text and no values are provided in the **Prompt Value** dialog box, a `Where` clause is usually generated using the default value. Ensure that the text you provide results in a valid SQL statement.

Parameter	Description
Text (optional)	Text that precedes any user-provided values, such as, 'and column1 = '.
QueryItem (optional)	The prompt engine can take advantage of the **Prompt Info** properties of the query item. Descriptive information can be shown although the prompt value is a code.
TextFollowing (optional)	The closing parenthesis which is used most often for the `promptmany` function. It is also useful when the prompt is optional and is followed by hardcoded filters in the SQL statement.

Examples - Select a Country Prompt

When a report is run, you want your users to be prompted to choose the country for which they want to see data. The following code examples describe how you can use macros to create different prompts.

Mandatory Prompt with No Data Type Specified

Note the following:

- The `Datatype` argument is not specified. Therefore, it is a string, which is correct in this case.
- The `DefaultText` argument is not specified. Therefore, it is a mandatory prompt.

```
select
 COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
 COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
 COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
 COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
 gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
where COUNTRY_MULTILINGUAL.COUNTRY = #prompt('CountryName')#
```

Mandatory Prompt with the Data Type Specified

Note the following:

- This prompt requires a valid integer value as response.
- The `DefaultText` argument is not specified. Therefore, it is a mandatory prompt.

```
select
 COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
 COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
 COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
 COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
 gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
where COUNTRY_MULTILINGUAL.COUNTRY_CODE >
 #prompt('Starting CountryCode',
   'integer',
   '',
```

```
   '',
 '[gosales].[COUNTRY_MULTILINGUAL].[COUNTRY_CODE]')#
```

Optional Prompt and Mandatory Filter with the Data Type and Default Value Specified

Note the following:

- This prompt allows the user to supply a valid integer response.
- The `DefaultText` argument is specified. Therefore, the user may omit entering a value, in which case the value 10 is used. This makes it an optional prompt, but not an optional filter.

```
select
 COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
 COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
 COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
 COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
 gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
where COUNTRY_MULTILINGUAL.COUNTRY_CODE >
#prompt('Starting CountryCode',
   'integer',
   '10'
)#
```

Prompt That Appends Text to the Value

Note the following:

- The `DefaultText` argument is specified as a space character. In this case, the generated text is just the space character, which eliminates the `Where` clause from the query.
- The `Text` argument is specified, which is written into the generated SQL before the user-provided prompt value.

```
select
 COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
 COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
 COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
 COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
 gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
#prompt('Starting CountryCode',
  'integer',
  ' ',          // < = = this is a space
 'where COUNTRY_MULTILINGUAL.COUNTRY_CODE >'
)#
```

Syntax Substitution

Note the following:

- The `Datatype` argument is set to `token`, which means that the user-provided value is entered into the SQL without any checking on the provided value.
- `Token` should be used only if there is a list of pick-values for the user.
- The `DefaultText` argument is specified. Therefore, this is an optional prompt and `group by COUNTRY` is used in the generated SQL.

```
Select
 COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
 COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
 COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
 COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
 gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
#prompt('Sort column',
   'token',
   'group by COUNTRY',
 'group by '
)#
```

Examples - Create a Prompt That Uses a Parameter Map

When a report is run, you want your users to select a language for the data in the report. The following examples describe several ways you can do this.

Prompt That Uses a Session Variable

Note the following:

- The name of the prompt is specified using a lookup in the parameter map named PromptLabels. The key value is the session variable $language.
- The `Where` clause is using a parameterized column.

```
select
 ORDER_METHOD.ORDER_METHOD_CODE as ORDER_METHOD_CODE,
 ORDER_METHOD.ORDER_METHOD_#$language#
 as ORDER_METHOD_EN
from
 gosales.gosales.dbo.ORDER_METHOD ORDER_METHOD
#prompt($PromptLabels{$language},
 '',
 ' ',
 'where ORDER_METHOD.ORDER_METHOD_' + $language + '
>'
)#
```

A Parameter Map That Nests Prompts

Note the following:

- In the model, there is a parameter map DynPromptLabels with `#$PromptLabels{$language}#`
- Part of the prompt information is run from a parameter map instead of being coded directly inside the SQL.
- The whole macro containing the prompt can be a value in a parameter map.

```
select
 ORDER_METHOD.ORDER_METHOD_CODE as ORDER_METHOD_CODE,
 ORDER_METHOD.ORDER_METHOD_#$language#
 as ORDER_METHOD_EN
from
 gosales.gosales.dbo.ORDER_METHOD ORDER_METHOD
#prompt($DynPromptLabels{'ex9'},
 '',
 ' ',
```

```
 'where ORDER_METHOD.ORDER_METHOD_' + $language + '
>'
)#
```

Examples - Create a Multiple Value Prompt

When a report is run, you want your users to select one or more values. The following examples describe several ways you can do this.

Prompt with a Required Minimum

Note the following:

- The user must enter at least a single value.
- This resembles the first example on prompting for a country (p. 171).

```
select
 COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
 COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
 COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
 COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
 gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
where COUNTRY_MULTILINGUAL.COUNTRY IN (#promptmany('CountryName')#)
```

Prompt with a Required Minimum with the Data Type Specified

Note the following:

- This resembles the second example on prompting for a country (p. 171).

```
select
 COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
 COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
 COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
 COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
 gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
where COUNTRY_MULTILINGUAL.COUNTRY_CODE   IN (
 #promptmany('Selected CountryCodes',
     'integer',
     '',
     '',
     '[gosales].[COUNTRY_MULTILINGUAL].[COUNTRY_CODE]')#
)
```

Optional Prompt with the Data Type and Default Value Specified

Note the following:

- The `In` clause and both parentheses are part of the SQL statement.

```
select
 COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
 COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
 COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
 COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
 gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
where COUNTRY_MULTILINGUAL.COUNTRY_CODE IN (
 #promptmany('Selected CountryCodes',
```

```
     'integer',
     '10'
 )#
)
```

Prompt That Adds Text Before the Syntax

Note the following:

- This example uses the `TextFollowing` argument.

```
select
 COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
 COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
 COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
 COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
 gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
#promptmany('Selected CountryCodes',
    'integer',
    ' ',     // < = = this
is a space
    'where COUNTRY_MULTILINGUAL.COUNTRY_CODE IN
( ',
    '',
    ')'
)#
```

Optional Prompt That Adds Text Before the Syntax

```
Select
 COUNTRY_MULTILINGUAL.COUNTRY_CODE as COUNTRY_CODE,
 COUNTRY_MULTILINGUAL.COUNTRY as COUNTRY,
 COUNTRY_MULTILINGUAL."LANGUAGE" as LANGUAGE1,
 COUNTRY_MULTILINGUAL.CURRENCY_NAME as CURRENCY_NAME
from
 gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL,
 gosales.gosales.dbo.COUNTRY XX
where COUNTRY_MULTILINGUAL.COUNTRY_CODE = XX.COUNTRY_CODE
 #promptmany('Selected CountryCodes',
     'integer',
     ' ',
  ' and COUNTRY_MULTILINGUAL.COUNTRY_CODE IN (',
     '',
     ')'
 )#
```

Organizing the Model

Information about organizing an SAP BW model appears in a different topic (p. 240).

When you organize the model, you make it easier for your users to find and understand the data in the model. You also make the model easier for you to manage and maintain. We recommend that you create several views, or layers, in the model:

- Keep the metadata from the data source in a separate namespace or folder.

  In Framework Manager, this is called the import view.

- Create one or more optional namespaces or folders for resolving complexities that affect querying using query subjects or dimensional objects.

To use Analysis Studio or any OLAP-style queries, there must be a namespace or folder in the model that represents the metadata with dimensional objects.

- Create one or more namespaces or folders for the augmented business view of the metadata that contains shortcuts to dimensions or query subjects.

  In Framework Manager, these are called the business view. Use business concepts to model the business view. One model can contain many business views, each suited to a different user group. You publish the business views.

Security can be defined in any of the views. It depends on your business requirements. For example, if you need to keep everyone from viewing an object, you add security to the object in the import view. Typically security is applied in the business view.

To organize the model, you can do the following:

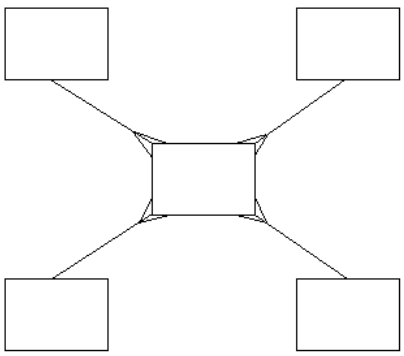
- create star schema groups (p. 176)
- include metadata in several folders by using shortcuts (p. 180)
- create namespaces or folders (p. 183)
- create query item folders (p. 184)
- create measure folders (p. 185)

Create a Star Schema Group

Use the **Star Schema Grouping** wizard to quickly create groups of shortcuts that will provide context for your users regarding which objects belong together. This makes the model more intuitive for your users. Star schema groups can also facilitate multiple-fact reporting by allowing the repetition of shared dimensions in different groups. This helps your users to see what different groups have in common and how they can do cross-functional, or multiple-fact, reporting. For more information, see "Multiple-fact, Multiple-grain Queries" (p. 327).

Star schema groups also provide context for queries with multiple join paths. By creating star schema groups in the business view of the model, you can clarify which join path to select when many are available. This is particularly useful for fact-less queries.

In a star schema design, numeric, transactional data is contained in a central fact table with related dimension tables radiating out from the fact table.



Star schema groups can contain the selected dimensions, query subjects, or shortcuts. The objects in a star schema group cannot reference, or be referenced by, any object outside the group. Therefore, Framework Manager automatically creates a shortcut for any object that is referenced by an object outside the group.

For example, in a project for sales analysis, you include these dimensions:

- dates of sales (Time)
- locations of sales (Region)
- product details (Product)
- customer information (Customer)

You include quantity in the fact table.

Steps

1. Select one or more dimensions or query subjects.
2. From the **Tools** menu, click **Create Star Schema Grouping**.
3. If you want to exclude an object from the group, in the **Available objects** box, clear the check box next to the object.
4. Do one of the following:
   - To add shortcuts to the group, click **Create a shortcut for all selected objects**.
   - To move the objects to the group, click **Create shortcuts only for objects that are used outside the star schema.**
5. To move the selected objects to a separate namespace, ensure that the **Create a new namespace for this grouping** check box is selected and type the name in the **Namespace name** box.
6. Click **OK**.
7. If there are multiple relationships, also known as role-playing dimensions, create relationship shortcuts for them (p. 85), or create individual dimensions or query subjects if you must rename them.

A Star Schema Group Based on One Dimension or Query Subject

Generally, you select a single object when it is a fact that has a relationship to every dimension or query subject that you want in the star schema group. When you create a star schema group that is based on one object, the following occurs:

- Framework Manager shows a list of objects with which it has relationships so that you can quickly select the objects that you want in the group.
- The name of the group is based on the name of the fact table.
- The new group is created under the same parent as the selected object.

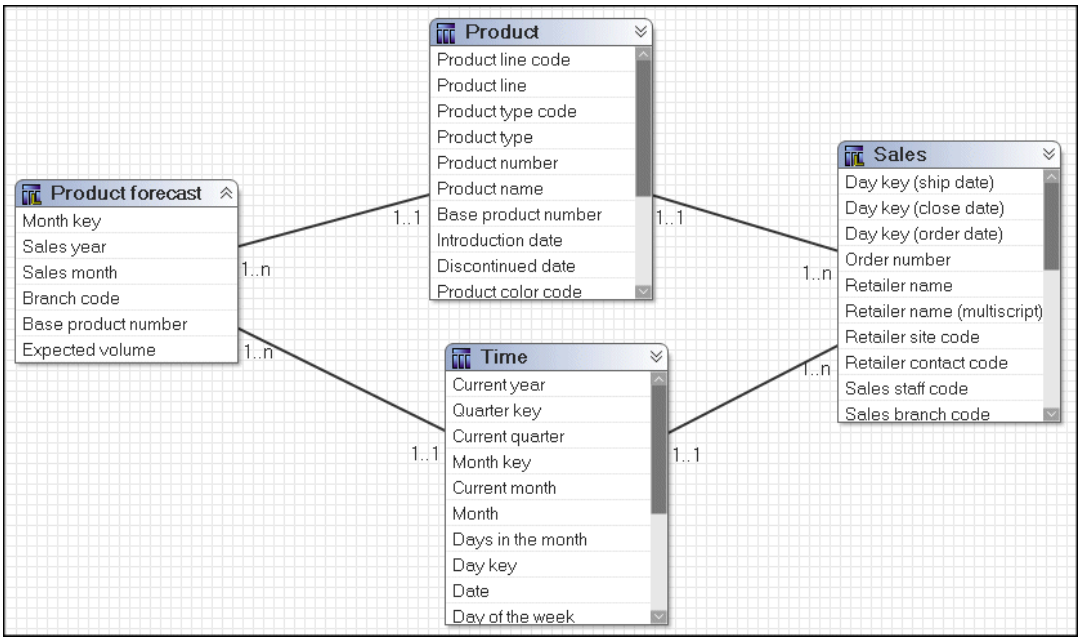
A Star Schema Group Based on Multiple Dimensions or Query Subjects

Selecting multiple dimensions or query subjects is useful if you want to group dimensions or query subjects that do not already have relationships defined. The new group is placed under the nearest common parent of the dimensions or query subjects.

Resolve Multiple Conformed Star Schemas

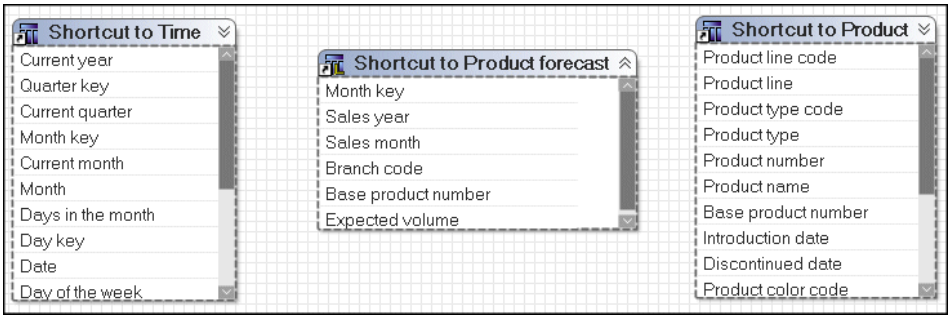
You will likely see dimensional query subjects that are joined to more than one fact query subject. Join ambiguity is an issue when you report using items from multiple dimensions or dimensional query subjects without including any items from the measure dimension or fact query subject. This is called a fact-less query.

For example, Product and Time are related to the Product forecast and Sales facts.



Using these relationships, how do you write a report that uses only items from Product and Time? The business question could be which products were forecasted for sale in 2005 or which products were actually sold in 2005. Although this query involves only Product and Time, these dimensions are related through multiple facts. There is no way to guess which business question is being asked. You must set the context for the fact-less query.

In this example, we recommend that you create two namespaces, one containing shortcuts to Product, Time, and Product forecast, and another containing Product, Time, and Sales.



When you do this for all star schemas, you resolve join ambiguity by placing shortcuts to the fact and all dimensions in a single namespace. The shortcuts for conformed dimensions in each namespace are identical and are references to the original object. **Note:** The exact same rule is applied to regular dimensions and measure dimensions.

With a namespace for each star schema, it is now clear to your users which items to use. To create a report on which products were actually sold in 2005, they use Product and Year from the Sales Namespace. The only relationship that is relevant in this context is the relationship between Product, Time, and Sales, and it is used to return the data.

Steps

1. Select one of the measure dimensions.
2. From the **Tools** menu, click **Create Star Schema Grouping**.
3. If you want to exclude an object from the group, in the **Available objects** box, clear the check box next to the object.
4. Click **Create a shortcut for all selected objects**.
5. To move the selected objects to a separate namespace, ensure that the **Create a new namespace for this grouping** check box is selected and type the name in the **Namespace name** box.
6. Click **OK**.
7. Repeat these steps for the other measure dimension.

Model a Snowflaked Dimension as a Star Dimension

A snowflaked dimension removes low-level cardinality attributes from the dimension tables and places them in secondary dimensions that are linked back to the original dimensions by artificial keys.

To model a snowflaked dimension as a star dimension, do the following:

- Select the query subjects that are required to access the data, and put them in a new namespace (p. 183).
- Ensure that all relationships are correct (p. 79).
- Handle multilingual metadata (p. 131).
- Create a model dimension for each snowflaked dimension:

- Select the query subjects you need.
- From the **Actions** menu, click **Merge in New Regular Dimension**.
- Rename the new model dimension. By default, its name is composed of the concatenated names of the original objects.

❑ If you require multiple levels, do one of the following:

- Use the dimension map to define hierarchies and levels for the dimension (p. 117).
- Specify determinants if the levels are stored in a single query subject (p. 94).

Use Shortcuts

Information about shortcuts for SAP BW metadata appears in a different topic (p. 240).

A shortcut is a pointer to an object, such as a relationship, a dimension, a query subject, or a folder. We recommend that you use shortcuts in the business view when there is an overlap between user groups and you want to include the metadata in more than one folder. With shortcuts, you can have multiple references to an object.

For example, you create folders named Orders, Products, and Customers. If you want both Orders and Customers to contain the same dimension, you must create a shortcut to the dimension and add it to both folders.

When you create a shortcut, Framework Manager does not set the **Screen Tip** and **Description** properties. Unless you define these properties, the values shown in the Cognos 8 studios are the same as those defined in the object that the shortcut references.

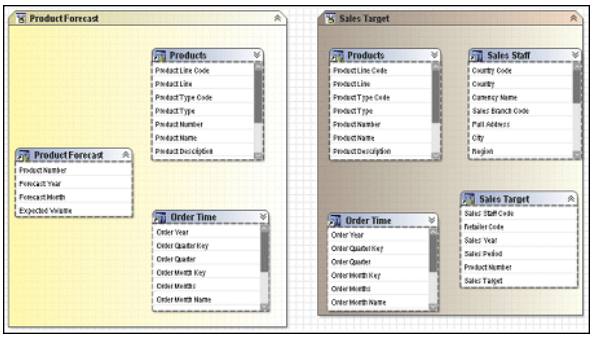
Tip: To go to the object that the shortcut references, right-click the shortcut and click **Go To Target**.

Shortcuts are less flexible from a presentation perspective than model objects, but they require much less maintenance because they are automatically updated when the target object is updated. If maintenance is a key concern and there is no need to customize the appearance of the query subject, use shortcuts.

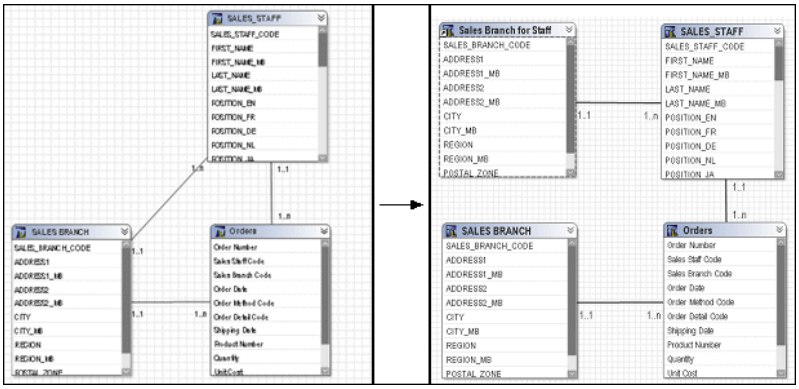
Framework Manager has two types of shortcuts:

- regular shortcuts, which are a simple reference to the target object.
- alias shortcuts, which behave as if they were a copy of the original object with completely independent behavior. Alias shortcuts are available only for query subjects and dimensions.

Regular shortcuts are typically used as conformed dimensions with star schema groups, creating multiple references with the exact same name and appearance in multiple places. In the example below, the shortcuts created for Products and Order Time behave as references. If a query is written that brings Products from both Product Forecast and Sales Target, the query uses the definition of Products based on the original and this definition appears only once in the query.



Alias shortcuts are typically used in role-playing dimensions or shared tables. Because there is already an example in this document for role-playing dimensions, we will look at the case of shared tables. In this example, Sales Staff and Sales Branch can be treated as different hierarchies. From our knowledge of the data, we know that because staff can move between branches, we need to be able to report orders against Sales Branch and Sales Staff independently as well as together. To achieve this, we need to create an alias to Sales Branch that can be used as a level in the Sales Staff hierarchy.



With the new alias shortcut in place, it is possible to create queries that require orders by sales branch and orders by sales staff with their current branch information simultaneously.

Shortcuts and Relationships

When you decide where to place shortcuts, consider how the scope of the shortcut affects relationships. Shortcuts in a different folder from the target query subject use the relationships of the target query subject. Shortcuts in the same folder as the target query subject ignore the relationships of the target query subject and use only those specified for the shortcut.

You can specify a different relationship for a shortcut than the relationships of the target query subject. By creating relationships from the shortcut to other query subjects, you avoid cross-join errors in the model.

You cannot create shortcuts to scope relationships.

Shortcuts and Dimensions or Query Subjects

Shortcuts result in fewer dimensions or query subjects to maintain. You can keep dimensions or query subjects in the import view and keep shortcuts in the business view. Shortcuts are most frequently used when creating star schema groups.

When you create a shortcut to a dimension or query subject, you cannot customize which query items are in the shortcut. The entire dimension or query subject is included in the shortcut.

Being able to specify the behavior of shortcuts is new to Cognos 8.3.

When you open a model from a previous release, the **Shortcut Processing** governor is set to **Automatic**. When **Automatic** is used, shortcuts work the same as in previous releases, that is, a shortcut that exists in the same folder as its target behaves as an alias, or independent instance, whereas a shortcut existing elsewhere in the model behaves as a reference to the original. To take advantage of the **Treat As** property, it is recommended that you verify the model and, when repairing, change the governor to **Explicit**. The repair operation changes all shortcuts to the correct value from the **Treat As** property based on the rules followed by the **Automatic** setting, this means that there should be no change in behavior of your model unless you choose to make one or more changes to the **Treat As** properties of your shortcuts.

When you create a new model, the **Shortcut Processing** governor is always set to **Explicit**.

When the governor is set to **Explicit**, the shortcut behavior is taken from the **Treat As** property and you have complete control over how shortcuts behave without being concerned about where in the model they are located.

Shortcuts can be created by the **Create Star Schema Grouping** wizard. For example, a fact table and its dimension tables are stored in the import view. If you want to represent conformed dimensions in several star schema groups, only one dimension or query subject can exist for each dimension table. Use shortcuts for all other instances of the dimension or query subject. By using shortcuts, you can build queries involving multiple fact tables that are related through shared dimension tables.

In the case of role-playing dominions, there are two main approaches documented in "Role-Playing Dimensions" (p. 339). You can create a query subject and regular dimension for each role and then use shortcuts as references in star schema groups to allow the use of the role-playing dimensions as conformed dimensions, or you can create a shortcut for each role in each star schema group and create a new join between each role-playing shortcut and its related fact shortcut. As long as the target of the shortcuts is not contained in the same folder, all shortcuts behave as aliases.

The security you specify for an object is passed to shortcuts that reference the secured object. If you have a shortcut to a secured object, only users with permission to see the secured object can see the shortcut in the published package.

If a shortcut is to a dimension or query subject, you can specify the behavior for the shortcut in the **Treat As** property. You can set the property to one of the following:

- Reference

  Use when you want an exact replica of a query subject in several places to behave as one object if referenced in the same query.

- Alias

  Use when you want an exact replica of a query subject to behave as an independent object that follows an independent join path. Independent join paths must first be defined in the model.

The **Shortcut Processing** governor controls the behavior for all shortcuts. The **Shortcut Processing** governor takes priority over the **Treat As** property. For example, if the governor is set to **Automatic**,

the behavior of the shortcut is determined by the location of the shortcut relative to its target no matter what the setting of the **Treat As** property. For more information, see "Set Governors" (p. 310).

Steps

1. Right-click the query subjects, dimensions, or folders that you want to create shortcuts to, and do one of the following:
   - Click **Create Alias Shortcut**.
   - Click **Create Shortcut**.
   - Click **Create Star Schema Grouping**. This command is also available from the **Tools** menu.
2. For shortcuts to query subjects or dimensions, in the **Properties** pane, set the **Treat As** property to **Alias** or **Reference**.

Create a Folder or Namespace

Information about folders and namespaces for SAP BW metadata appears in a different topic (p. 242).

You can create folders or namespaces to organize objects in the model.

The most important thing to know about namespaces is that once you have begun authoring reports, any changes you make to the names of published namespaces will impact your Cognos 8 content. This is because changing the name of the namespace changes the IDs of the objects published in the metadata. Because the namespace is used as part of the object ID in Cognos 8, each namespace must have a unique name in the model. Each object in a namespace must also have a unique name. Part of the strategy of star schema groups is placing shortcuts into a separate namespace, which automatically creates a unique ID for each object in the namespace. This allows us to use the same name for shortcuts to conformed dimensions in different star schema groups.

The next time you try to run a query, report, or analysis against the updated model, you get an error. If you need to rename the namespace that you have published, use **Analyze Publish Impact** to determine which reports are impacted.

Folders are much simpler than namespaces. They are purely for organizational purposes and do not impact object IDs or your Cognos 8 content. You can create folders to organize objects by subject or functional area. This makes it easier for you to locate metadata, particularly in large projects.

The main drawback of folders is that they require unique names for all query subjects, dimensions, and shortcuts. Therefore, they are not ideal for containing shared objects such as conformed dimensions.

Tip: When viewing metadata in the **Diagram** tab, you can expand or collapse folders and namespaces. From the **Diagram** menu, click **Collapse All** or **Expand All**.

If you set security on a folder and then move objects into the folder, confirm that exclusions are set correctly.

Steps to Create a Namespace

1. From the **Actions** menu, click **Create**, **Namespace**.
2. Right-click the namespace, click **Rename**, and give the namespace a descriptive, unique name.
3. Add objects by importing metadata or moving model objects or shortcuts to the objects into the namespace.

Steps to Create a Folder

1. From the **Actions** menu, click **Create**, **Folder**.
2. In the **Folder name** box, type a name for the new folder.
3. Click **Next**.
4. Choose whether to move the objects or to create shortcuts:
   - To move selected objects to the folder, click **Move the selected items to the new folder**. When you move an object that participates in a relationship, the relationships to this object also move.
   - To create shortcuts that reference selected objects, click **Create a shortcut for the selected items**. Do not select all the objects in the namespace to avoid creating a recursive structure in the published package.
5. Select the objects you want to add to the folder.
6. Click **Finish**.

Create a Query Item Folder

You can create query item folders to organize query subjects or dimensions that contain a large number of query items. A query item folder can contain only query items and query item folders.

If you create a query item folder, you cannot change the order of query items in the **Edit Definition** dialog box for the model query subject. You can change the order of items only in the **Project Viewer**.

Steps

1. In the **Project Viewer** pane, click a query subject or dimension.
2. From the **Actions** menu, click **Create**, **Query Item Folder**.

   A new query item folder appears in the **Project Viewer**, under the query items that belong to that query subject or dimension.
3. Drag the query items that you want into the query item folder.

You cannot add query items that do not exist in the parent query subject or dimension.

Create a Measure Folder

You can create measure folders to organize measure dimensions that contain a large number of query items. You can nest measure folders within other measure folders.

You cannot create a measure folder from a measure shortcut.

If you create a measure folder, you cannot change the order of measures in the **Edit Definition** dialog box for the measure dimension. You can change the order of measures only in the **Project Viewer**.

Steps

1. In the **Project Viewer** pane, click a measure dimension.
2. From the **Actions** menu, click **Create**, **Measure Folder**.

   A new folder appears in the **Project Viewer**, under the measures that belong to that measure dimension.
3. Drag the query items that you want into the measure folder.

   You cannot add measures that do not exist in the parent measure dimension.

Analyze a Model

You can analyze the metadata in a model by using the **Model Advisor**, which is an automated tool that applies rules based on current modeling guidelines and identifies areas of the model that you need to examine. To assist you in understanding the nature of the highlighted issue as well as some possible actions, you are provided with links to the appropriate sections of the documentation. The **Model Advisor** is not a replacement for a knowledgeable modeler; it provides new modelers with an assistive tool and more experienced modelers with a diagnostic tool.

You can select one or more tests to run against the selected model or subset of a model. We recommend that you verify the model and fix errors before analyzing the model (p. 247).

If you are analyzing a new model, we recommend the following workflow:

- Analyze newly-imported objects, especially their relationships and determinants.
- Use the issues that are identified to resolve potential query generation issues.
- As you build additional views, use the **Model Advisor** to analyze each one for potential issues.
- Before publishing the model, use the **Model Advisor** on objects that will be published.

If you are analyzing an older, established model or a model that is not yet complete, use the **Model Advisor** to validate modeling practices. The workflow is similar to that used for new models: start at the database view and work up.

Facts Identified by Cardinality

This test looks for query subjects that have only the many (n) cardinality on the ends of the relationship in which they are involved. Query subjects with this cardinality are treated as facts when generating queries so it is important to ensure that they are correctly identified.

Query Subjects That Can Behave as Facts or Dimensions

This test looks for query subjects that have a combination of the many (n) and one or zero (1,0) cardinality. Mixed cardinality means that the behavior of a query subject can change depending on the other query subjects used in a query. This can lead to unpredictable queries in some cases. If the query subject is evaluated as a fact, it will be included in the query path. If it is evaluated as a dimension, it may be skipped if it is not directly referenced in the query. In the interests of ensuring predictable behavior, it is recommended. that you resolve these scenarios.

Mixed cardinality can indicate a query subject that is part of a snowflaked dimension or master-detail relationship. In the case of intermediate tables in snowflaked dimensions, there is no issue to resolve. Problems may arise in situations where there are multiple query paths available; these cases should be highlighted by the test for query subjects with multiple relationships.

Query Subjects with Multiple Relationships

This test looks for query subjects that have either many relationships between two objects or a loop join that does not represent the star schema join pattern.

Multiple relationships between two query subjects is often associated with role-playing dimensions. With role-playing dimensions, you create aliases that enable each role to have a different join and different independent behavior. Multiple join paths that are indicative of loop joins (except for star schemas) may lead to problems of incorrectly split queries. This impacts the predictability of query generation for your users.

If Cognos 8 has multiple relationships with no distinguishing criteria to choose from, it uses the relationship that comes first alphabetically. If you need to create a query that uses a different relationship, you always have a problem. As well, if you want to use filters on criteria that are specific to the role defined by the relationships and those filters are mutually exclusive, then no data is shown in the report. For information about role-playing dimensions and an example, see "Role-Playing Dimensions" (p. 339).

Note: When cardinality clearly identifies facts and dimensions, Cognos 8 can automatically resolve loop joins that are caused by star schema data when you have multiple fact tables joined to a common set of dimension tables. The **Model Advisor** ignores star schema join patterns because they do not qualify as problematic joins.

Query Subjects That Join to Themselves

This test looks for reflexive and recursive relationships. These relationships imply two or more levels of granularity. Framework Manager imports reflexive relationships but does not use them when executing queries. Reflexive relationships, also called self-joins, are shown in the model for the purpose of representation only.

Most often a query subject that joins to itself indicates a parent-child relationship. Data sets that have parent-child relationships can be of definite or indefinite depth. The only way to know which you are dealing with is to understand the data and the business concept represented.

Whether you have a problem when you encounter a parent-child relationship depends on how you intend to use the data in your application. There may be cases where only one level of the relationship is necessary, and this is a fairly simple scenario to model. If you know the data to be fully populated for each level and to have a definite number of levels, you can also choose a modeling-only solution. However, for cases where there is a significant amount of data, the data is not fully populated at each level (ragged or unbalanced hierarchy) or the number of levels in the data could change over time or both, it is recommended that you transform the data to a flat structure with a fixed number of columns before modeling in Framework Manager.

Determinants That Conflict with Relationships

This test looks for determinants that conflict with the relationship defined for the query subject. Determinants are used to ensure that the aggregation in reports is correct and that queries generate correctly.

Determinants reflect granularity by representing subsets or groups of data in a query subject and are used to ensure correct aggregation of this repeated data. Determinants are most closely related to the concept of keys and indexes in the data source and are imported based on unique key and index information in the data source. We recommend that you always review the determinants that are imported and, if necessary, modify them or create additional ones. By modifying determinants, you can override the index and key information in your data source, replacing it with information that is better aligned with your reporting and analysis needs. By adding determinants, you can represent groups of repeated data that are relevant for your application.

The **Model Advisor** checks query subjects with determinants and flags the ones that meet one of the following criteria:

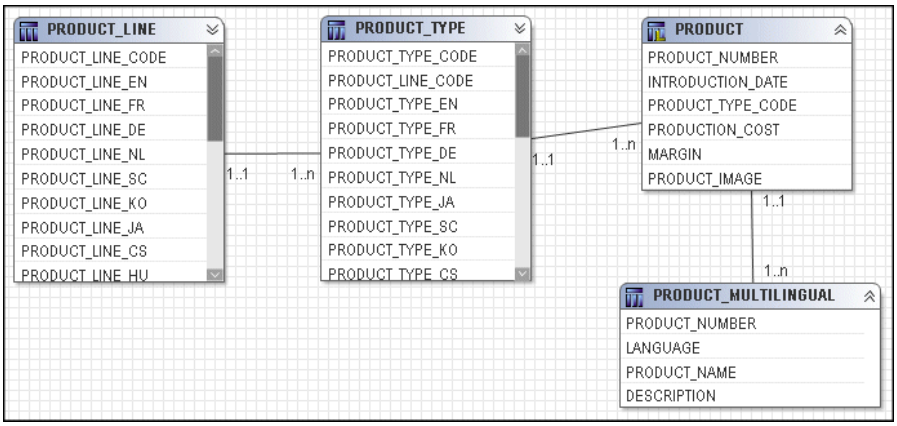
- the relationship references all query items in the determinant and the determinant uniquely identifies a row but the cardinality of the relationship is not `1:1` or `0:1`
- the relationship does not reference all query items in the determinant and the determinant uniquely identifies a row but the cardinality of the relationship is not `1:n` or `0:n`

Factors That Will Override the Minimized SQL Setting

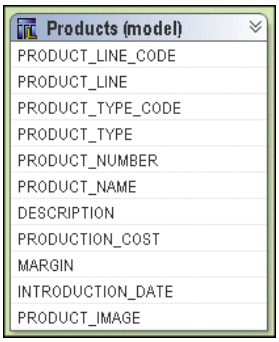
This test looks for various factors that override the SQL Generation type setting of Minimized, such as modified data source query subjects, relationships between model query subjects, or determinants for model query subjects.

When you use minimized SQL, the generated SQL contains only the minimal set of tables and joins needed to obtain values for the selected query items.

To see an example of what minimized SQL means, here are four query subjects, Product Line, Product Type, Product, and Product Multilingual that join to each other.



They can be combined in a model query subject.



For example, if you test the Products (model) query subject as a whole, you see that four tables are referenced in the `from` clause of the query.

```
Cognos SQL
select
       PRODUCT_LINE.PRODUCT_LINE_CODE  as  Product_Line_Code,
       PRODUCT_LINE.PRODUCT_LINE_EN  as  Product_Line,
       PRODUCT_TYPE.PRODUCT_TYPE_CODE  as  Product_Type_Code,
       PRODUCT_TYPE.PRODUCT_TYPE_EN  as  Product_Type,
       PRODUCT.PRODUCT_NUMBER  as  Product_Number,
       PRODUCT_MULTILINGUAL.PRODUCT_NAME  as  Product_Name,
       PRODUCT_MULTILINGUAL.DESCRIPTION  as  Product_Description,
       PRODUCT.INTRODUCTION_DATE  as  Introduction_Date,
       PRODUCT.PRODUCT_IMAGE  as  Product_Image,
       PRODUCT.PRODUCTION_COST  as  Production_Cost,
       PRODUCT.MARGIN  as  Margin
from
       gosl_82..gosl.PRODUCT_LINE PRODUCT_LINE,
       gosl_82..gosl.PRODUCT_TYPE PRODUCT_TYPE,
       gosl_82..gosl.PRODUCT PRODUCT,
       gosl_82..gosl.PRODUCT_MULTILINGUAL PRODUCT_MULTILINGUAL
where
       (PRODUCT_MULTILINGUAL."LANGUAGE" = N'EN') and
       (PRODUCT_LINE.PRODUCT_LINE_CODE = PRODUCT_TYPE.PRODUCT_LINE_CODE) and
       (PRODUCT_TYPE.PRODUCT_TYPE_CODE = PRODUCT.PRODUCT_TYPE_CODE) and
       (PRODUCT.PRODUCT_NUMBER = PRODUCT_MULTILINGUAL.PRODUCT_NUMBER)
```

If you test only Product Name, you see that the resulting query uses only Product Multilingual, which is the table that was required. This is the effect of minimized SQL.

```
Cognos SQL
select
       PRODUCT_MULTILINGUAL.PRODUCT_NAME  as  Product_Name
from
       gosl_82..gosl.PRODUCT_MULTILINGUAL PRODUCT_MULTILINGUAL
where
       (PRODUCT_MULTILINGUAL."LANGUAGE" = N'EN')
```

Embedded Calculations That Use the Calculated Aggregation Type

This test detects where you have set the **Regular Aggregate** property to **calculated** for a measure in a measure dimension. The **calculated** aggregation type is only supported for stand-alone calculations at this time and does not yield expected results if it is embedded in a measure dimension.

Query Subjects That Can Cause a Metadata Caching Conflict

This test looks for factors that override cached metadata, such as data source query subjects whose SQL has been modified or query subjects that contain calculations or filters.

Framework Manager stores the metadata that is imported from the data source. However depending on the governor settings and certain actions you take in the model, this metadata might not be used when preparing a query. If you select the **Allow enhanced model portability at run time** governor, Framework Manager always queries the data source for information about the metadata before preparing a query. If you have not selected the **Allow enhanced model portability at run time** governor, Framework Manager accesses the metadata that has been stored in the model instead of querying the data source. There are exceptions and the main cases are:

- Any modification of the SQL in a data source query subject. This includes the use of macros.
- Adding a calculation or filter to a data source query subject.

Note: The metadata queries generated by Cognos 8 are well supported by most relational database management system vendors and should not have a noticeable impact on most reporting applications.

Constraints

The **Model Advisor** is only intended for use with relationally-based metadata models. We do not recommend running the **Model Advisor** against an entire model; instead apply it to specific views one at a time to ensure that the feedback is being taken in context. For example, if an issue in an import view has not been addressed, the issue might be resolved by modeling that is done in an intermediate view.

For large models or namespaces, the **Model Advisor** might not return results immediately.

Because the **Model Advisor** is not data sensitive, you must know the data and model the metadata appropriately for your business intelligence needs. Not all items flagged by the **Model Advisor** are indicative of a problem. The context of each issue raised by the **Model Advisor** is important.

Steps

1. Click one or more objects to analyze.
   - Select query subjects, dimensions, hierarchies, calculation, query items, or shortcuts to analyze objects that will appear on a specific report to test the report before it is created.
   - Select a folder or namespace to analyze all its objects. If an object references an object in another folder or namespace, the referenced object is also analyzed.
   - Select a package before publishing it to ensure it follows the recommended guidelines for modeling.
2. From the **Tools** menu, click **Run Model Advisor**.

   Tip: You can also right-click one or more objects and then click **Run Model Advisor**.
3. In the **Options** tab, select the criteria that you want to use in the analysis.
4. Click **Analyze**.

5. In the **Model Advisor** tab, review the issues that are identified.

   There is a description of each issue, a link to more information about each issue, and a list of objects that are impacted by the issue.

6. To understand whether there is a problem with an object, click the **context explorer** button in the **Action** column of the report.

   The **Context Explorer** shows the objects that the selected object is connected to. You can selected a related object and see which objects it is connected to.

Chapter 5: Working with SAP BW Metadata

Note: Information on relational metadata is in "Modeling Relational Metadata" (p. 79).

After importing metadata, you must ensure that it is set up to meet your users' reporting requirements, and provide any additional information that they require. Enhancements you make in Framework Manager do not affect the original data source.

Tip: To verify that the model meets the reporting requirements, you can select objects that will appear in a report and test them. The test results show you the report that your users will see as well as the SQL and messages from Cognos 8, if any. Or you can publish a package at any time and then use the package to create reports.

You can check the project at any time to ensure that the references between the objects it contains are valid (p. 247).

You can do the following when working with SAP BW metadata in Framework Manager:

- Import the metadata (p. 191).
- Work with dimensions (p. 198).
- Control how data is used and formatted by checking query item properties (p. 213).
- If required, add more business rules, such as calculations and filters, to refine the retrieved data and to ensure that the right information is available for your users (p. 231).
- Organize the model by creating separate views for each user group that reflect the business concepts familiar to your users (p. 240).
- If required, adjust settings in Framework Manager and the Cognos 8 studios to optimize performance (p. 243).

After working with the model, you can create a package and publish it for your users. For more information, see "Publish a Package" (p. 261).

Import from an SAP BW Data Source

When you import from an SAP BW data source, you can import all the metadata or import only the objects you select. For information about mapping SAP BW metadata objects to Framework Manager objects, see "Mapping SAP BW Objects to Framework Manager" (p. 197).

You may want to have different views (or layers) in the model: an import view to contain the metadata you imported from the data source and a business view where you enhance the metadata. After importing, you can copy the metadata to the business view. You then have two views that must be synchronized with the BW InfoProvider.

Tip: If you want to expose calculated key figures from a SAP BW Query, you must import the SAP BW Query (p. 196).

Access to SAP BW Metadata and Data

When using an SAP BW data source, users' access to an InfoCube or InfoQuery metadata does not imply that they also have access to data within those objects. To enable Framework Manager to retrieve metadata from SAP BW, access privileges must be set up within the SAP BW system. To ensure that users have proper access permissions, verify the permissions assigned to the users' roles.

The following authorization objects must be configured so that Framework Manager can import information cubes or data sources, known as InfoCubes in the SAP BW system.

Authorization object	Field	Value
S_RFC	Activity	16 - Execute
	Name of RFC to be protected	SYST, RSOB, SUGU, RFC1, RS_UNIFICATION, RSAB, SDTX
	Type of RFC object to be protected	FUGR - Function group
S_RS_ICUBE	Activity	3 - Display
	InfoCube sub-object	DATA
	InfoArea	*InfoArea_of_the_ InfoCube*
	InfoCube	*Name_of_ InfoCube*
S_RS_HIER	Activity	71 - Analyze
	Hierarchy Name	*Name_of_Hierarchy*
	InfoObject	*Name_of_InfoObject*
	Version	*Hierarchy_Version*
S_TABU_DIS	Activity	03 - Display
	Authorization Group	&NC&

Tips:

- `&NC&` represents any table that does not have an authorization group. For security reasons, create a new authorization group and assign the table RSHIEDIR to it. The new authorization group restricts the user's access to the above table only, which is needed by Framework Manager. Create the new authorization group as a customization in the SAP BW system.

- You can use the asterisk (*) to represent all values, when it appears alone, or partial values, when used anywhere in a string.

SAP BW Hierarchies

When importing metadata, Framework Manager generates a dimension in each SAP BW characteristic.

Only one hierarchy associated with a given characteristic can be used in a report. Therefore, you should group dimensions that represent the hierarchies of a single characteristic into a folder or model query subject to make reporting easier for your users.

If there are multiple hierarchies in an SAP BW data source, the first hierarchy that is imported becomes the default hierarchy.

Framework Manager supports the following types of hierarchies:

- characteristic

  This is a list of all the characteristic values.

- text node

  Non-leaf nodes contain only text and do not reference any other data source object.

- characteristic value

  The nodes of each level of a presentation hierarchy are values from another characteristic.

- recursive

  The nodes of the entire presentation hierarchy are from the characteristic itself.

If a characteristic is not in a time dimension but it is a date and is treated as a date in SAP BW, the characteristic is imported with the date data type.

Framework Manager does not support hierarchies that contain two or more types of nodes. These hierarchies are imported but are hidden in the Framework Manager model.

Because hierarchical metadata is automatically generated for SAP BW, you cannot change it in Framework Manager.

Versioned Hierarchies

You can import the following types of versioned hierarchies from an SAP BW data source:

- Version dependent hierarchy

  A hierarchy can have multiple versions. Each version of a hierarchy can have a different structure, such as Sales by Region and Sales by Manager. During metadata import, Framework Manager identifies each version as a separate hierarchy and creates a dimension for each.

- Entire hierarchy time dependent

  Each version has an associated time period that does not overlap with any other version of the same hierarchy. The structure of each version can be different. During metadata import, Framework Manager identifies each version as a hierarchy and includes the applicable time period as part of the dimension name.

- Time-dependent hierarchy structure

  There is a single version of the hierarchy, but nodes within the hierarchy can be assigned applicable time periods. Over time, the structure of the hierarchy can change with new levels being introduced or removed. For example, levels that represent different sales districts can be added over time. During metadata import, Framework Manager identifies a time-dependent hierarchy structure as a non-versioned hierarchy and recognizes the structure of the hierarchy as of the current date.

The type of dimension determines which hierarchy is used and, for time-dependent hierarchies, which date to use to control the version.

Framework Manager sets the query key date of time-dependent hierarchies based on dates that are contained within the time-dependent hierarchy. You can then select specific versions of hierarchies. For hierarchies with versions on time, the default is the current date and time. The hierarchy that you apply to a characteristic depends upon the type of query key date: fixed, current, or variable. The query key date is set for a specific date.

For fixed date, include only the version that corresponds to the fixed date in the underlying SAP BW Query. For example, if the SAP BW Query has a fixed date such as 2005, only 2005 is imported.

For current date, include only the version that encompasses a time span appropriate for the present until some reasonable time in the future.

For variable, set the date for the variable in Framework Manager and include only the version of the hierarchy applicable to that date.

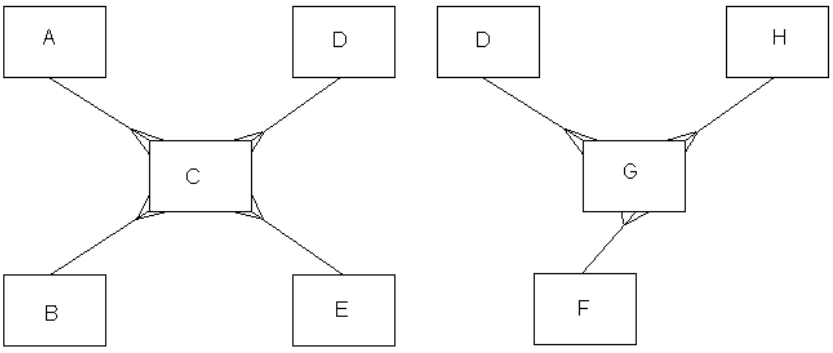
When you use Framework Manager to model SAP BW data, any versions or dates applied to a presentation hierarchy in SAP BW are not imported into the model. Therefore, all versions of the hierarchy are accessible in Framework Manager.

You may have a time-dependant hierarchy and a variable defined in SAP BW to establish the effective date for the hierarchy. In this case, assign a fixed date to the variable in Framework Manager and include only the dimension that corresponds to that date in the model.

In Framework Manager, if a versioned hierarchy is not time-dependent and has a fixed version, include only the version of the hierarchy associated with the selected version. Otherwise your users are presented with a hierarchy that is inaccessible.

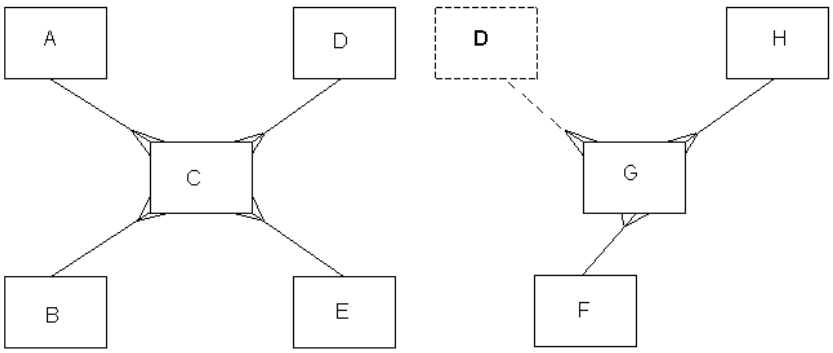
Conformed Dimensions

When you import multiple SAP BW Queries or InfoCubes, there can be two or more hierarchies that contain the same information.



You can conform these hierarchies in Framework Manager to create one dimension and one or more shortcuts. These hierarchies must have the same levels and properties. However, the order of the properties can be different. By conforming dimensions, your users can create reports that use multiple cubes without using an SAP BW MultiProvider.

A conformed dimension is represented by a dimension with one or more shortcuts to it. The dimension contains a list of data sources in which the dimension exists. If you want to retrieve data from a specific data source, ensure that you import the dimension from this data source before all others.



The dotted line in the diagram indicates a shortcut.

After import, you can also use Framework Manager to conform dimensions within a project. For more information, see "Conform SAP BW Dimensions" (p. 205).

SAP BW Structures

Many existing SAP BW queries contain structures that you can use in Cognos 8 queries to control the amount and order of information that your users see. For example, with dual structures, you can create a crosstab report with one structure on each axis.

The structures are:

- key figure structure

  The SAP BW Query Designer automatically creates a key figure structure when you add key figures to a query. You must have at least one key figure to import the query metadata into Framework Manager. This is true even when you do not use the key figure in reports. Therefore, you will always have a key figure structure.

- characteristic structure

  A characteristic structure is a collection of characteristic values (members) from one or more dimensions. You create a characteristic structure in SAP by adding a structure to the query, and then adding the required members to the structure. In Cognos 8, the structure appears as an additional dimension that has only one multiple-root level.

When you import the SAP BW query into Framework Manager, the key figure structure appears in the measure dimension called Key Figures and the characteristic structure appears as an additional dimension.

If you re-import the same SAP BW query into Framework Manager, you must use the same setting for the **SAP BW Dual Structures Support** check box. Framework Manager does not allow you to

select a different setting for the same query because different objects are then generated in the model and this leads to errors. You can use different settings for different queries.

In earlier releases of Cognos 8, dual structures imported into Framework Manager appeared as a measure dimension whose members were the cross product of the two structures. The key figure structure appeared as another dimension not as a measure dimension.

Steps to Access a Secured InfoCube

1. Create a query in Business Explorer Query Designer that accesses the InfoCube.
2. Create an authorization variable for each InfoObject in the underlying InfoCube for which there are authorizations.
3. For each variable, ensure that the Ready for Input option is disabled.

   By default, this option is enabled.
4. Enable the query for access through OLE DB for OLAP.
5. Save the query.
6. In Framework Manager, reference the query instead of the InfoCube.

Steps to Import from an SAP BW Data Source

1. Ensure that there is a connection to the data source (p. 61).
2. Click the namespace, folder, or segment you want to import into and from the **Actions** menu, click **Run Metadata Wizard**.
3. Select a data source connection and click **Next**.
4. Select the objects you want to import.

   If you are re-importing, the existing object gets updated.

   If you are importing new objects and an object with the same name exists, the new object is imported and a number is appended to the original name. For example, you see QuerySubject and QuerySubject1 in your project.

   After they are imported, query items cannot be deleted without deleting the entire query subject.
5. If you want to import a characteristic structure and a key figure structure, select the **SAP BW Dual Structures Support** check box.

   The content of the **Select Objects** page is updated to reflect the dual structures in your data source.

   Note: If you are re-importing the same SAP BW query (for example, because the underlying data source has changed), you must use the same setting for this check box. Framework Manager does not allow you to select a different setting for the same query because different objects are then generated in the model and this leads to errors.
6. Select the languages you want to import.

These languages must exist in the data source.

You can add languages to your project later, but you cannot go back and import the language-specific metadata from the data source. After the import is complete, the language-specific metadata must be added manually.

7. Indicate whether you want Framework Manager to show the short name, long name or the technical name for the dimensions.

   If you select short name and the field is empty, the long name is shown. If you select the long name and the field is empty, the short name is shown. If you select either short name or long name and both fields are empty, the technical name is shown.

8. To have objects in the model organized the same way that they are organized in Business Explorer Query Designer, select the **Enhance model for SAP BW organization of objects** option.

   You will then have a folder for each characteristic.

9. Indicate whether you want to generate conformed dimensions.

10. Click **Next**.

    A list of objects that could not be imported appears along with counts of objects that were imported.

11. Click **Finish**.

After importing, you should check the usage and aggregation property values (p. 216). Fact tables can contain numeric columns that should not be aggregated, such as exchange rates.

When you want to recreate a query on another SAP BW system, use the SAP BW migration mechanism to transport the query. This ensures that the technical name of each measure remains the same so that any project that references the query can be directed to either system without any modifications to the project.

Mapping SAP BW Objects to Framework Manager

SAP BW objects are mapped to the following Framework Manager objects.

SAP BW object	Framework Manager object
Query, InfoCube, RemoteCube, MultiCube	Namespace.

SAP BW object	Framework Manager object
Characteristic	A folder that contains dimensions. You must select the **Enhance model for SAP BW organization of objects** option when importing metadata to have a folder for each characteristic. **Note:** By default, Framework Manager imports SAP BW Currency and Unit of Measure characteristics. You can remove these characteristics if you do not need them.
Dimension	Dimension. The dimension may contain hierarchies representing each presentation hierarchy. The default hierarchy contains two levels representing • the aggregation of all characteristic values, also known as the All value • all characteristic values
Key figure	Query item that is part of a measure dimension called Key Figures.
Presentation hierarchy level	Level. **Note:** Level names must be defined in the Administrator Workbench to be meaningful.
Attribute	Query item associated with a level whose **Usage** property value is set to **Attribute**.
SAP BW variable	Data source property. For information about the SAP BW variables that Framework Manager supports, see "SAP BW Variables" (p. 223).

For information about setting access privileges to retrieve metadata from SAP BW, see "Using the Expression Editor" (p. 387).

Dimensions

Information about dimensions based on relational metadata appears in a different topic (p. 115).

A dimension is a broad grouping of data about a major aspect of a business, such as products, dates, or markets.

The types of dimensions that you can work with in Framework Manager are regular dimensions and measure dimensions. In SAP BW, measure dimensions are called key figures.

For example, in a project for sales analysis, you include these dimensions:

Name	Type	Description
Time	Regular dimension	Dates of sales organized into years, quarters, months, weeks, and days when sales were made
Region	Regular dimension	Locations of sales grouped into sales regions, countries, and cities
Product	Regular dimension	Product details organized by product type, brand, model, color, and packaging
Customer	Regular dimension	Customer information
Sales	Key Figures dimension	Purchase details such as units sold, revenue, and profit

Modify a Regular Dimension

Information about creating a regular dimension for relational metadata appears in a different topic (p. 117).

A regular dimension contains descriptive and business key information and organizes the information in a hierarchy, from the highest level of granularity to the lowest. It usually has multiple levels and each level requires a key and a caption. If you do not have a single key for your level, it is recommended that you create one in a calculation.

Model regular dimensions are based on data source or model query subjects that are already defined in the model. You must define a business key and a string type caption for each level. When you verify the model, the absence of business keys and caption information is detected. Instead of joining model regular dimensions to measure dimensions, create joins on the underlying query subjects and create a scope relationship between the regular dimension and the measure dimension.

When dimensions are based on SAP BW metadata, you cannot edit the underlying query.

Steps

1. Click the regular dimension you want to modify.
2. From the **Actions** menu, click **Edit Definition**.
3. Choose the action that you want:
   - Embed calculations by selecting the level, clicking **Add**, and then defining the expression (p. 231).
   - Embed filters (p. 233).
   - Test the dimension (p. 208).

- Conform (p. 205) or remove conformed dimensions (p. 207).

4. Click **OK**.

Hierarchies for a Regular Dimension

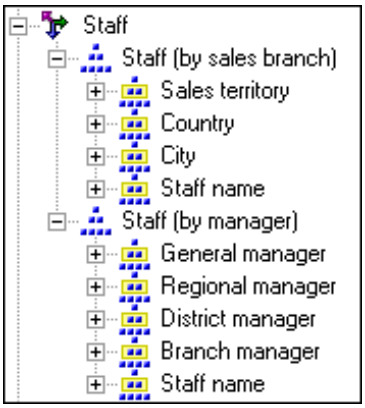
Information about hierarchies for relational metadata appears in a different topic (p. 119).

A hierarchy is an ordered list of levels or a collection of items. Each query item in a hierarchy must have a unique name.

You can specify multiple hierarchies on regular dimensions in Framework Manager. Multiple hierarchies for a regular dimension behave as views of the same query. The first hierarchy is the primary or default hierarchy.

You can use only one hierarchy at a time in a query. For example, you cannot use one hierarchy in the rows of a crosstab report and another hierarchy from the same dimension in the columns. If you need both hierarchies in the same report, you must create two dimensions, one for each hierarchy. For more information, see "Modeling Dimensions with Multiple Hierarchies" (p. 345)

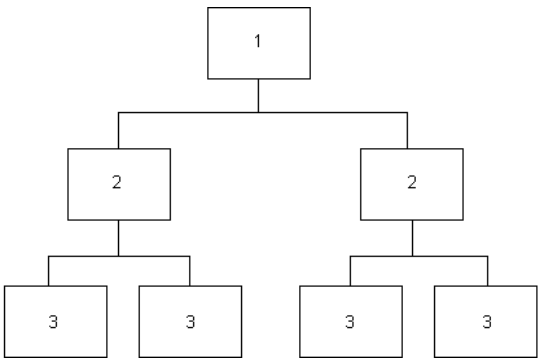
For example, sales staff can be viewed by manager or by geography and can be modeled as a single dimension with two hierarchies.



Balanced Hierarchy

Each path in a balanced hierarchy descends to the same depth.

For example, in the following diagram, the highest level is Product Line. Level 2 is Product Type. Level 3 is Products.



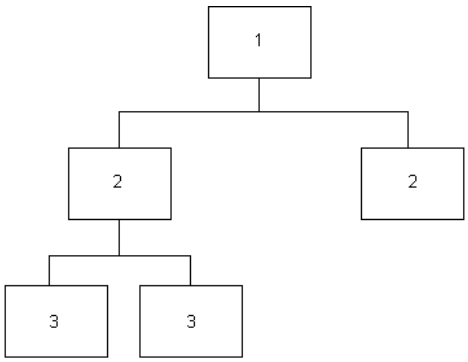
In SAP BW, all leaf nodes of a hierarchy are values of the characteristic, but each path does not have to descend to the lowest level of the hierarchy.

For information about optimizing performance for balanced hierarchies, see (p. 244).

Unbalanced Hierarchy

The branches in an unbalanced hierarchy descend to different levels.

For example, in the following diagram, the highest level in an organization is the CEO. Level 2 is the vice-presidents and the CEO's executive assistant. The executive assistant does not have subordinates although the vice-presidents do.



An unbalanced hierarchy can also be ragged. In a ragged-unbalanced hierarchy, there are gaps in the levels and the levels descend to different depths.

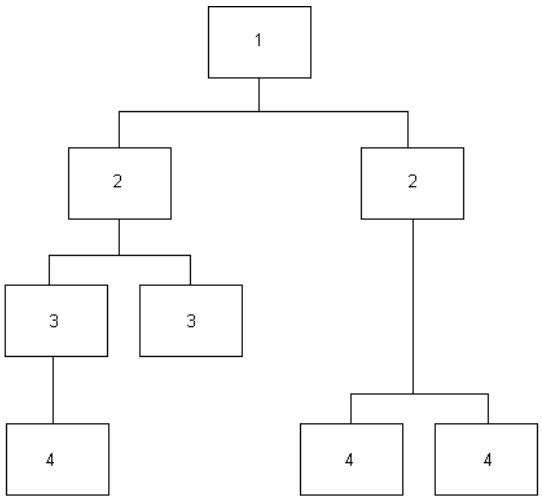
In SAP BW, this occurs only when there are "not assigned" (or "#") nodes in a presentation hierarchy. However, the presence of such a node does not guarantee that the hierarchy is unbalanced. You must investigate the layout of a hierarchy to be certain.

For information about optimizing performance for unbalanced hierarchies, see (p. 245).

Ragged Hierarchy

At least one path in the hierarchy skips at least one level.

For example, the highest level could be Continent. Level 2 could be Country. Level 3 could be City. Level 4 could be Street. A country such as Vatican City contains only streets, no cities.



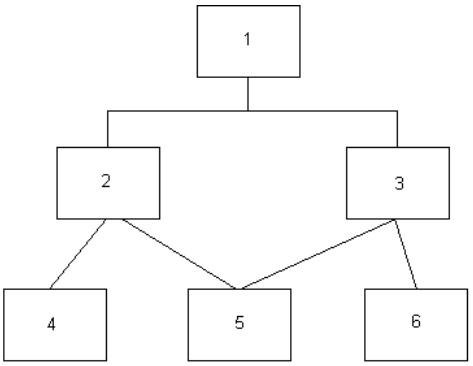
In SAP BW, this occurs only when there are not assigned or # nodes in a presentation hierarchy. However, the presence of such a node does not guarantee that the hierarchy is ragged. You must investigate the layout of a hierarchy to be certain.

For information about optimizing performance for ragged hierarchies, see (p. 244).

Network Hierarchy

A member of the hierarchy has more than one parent.

For example, an employee may report through different organizations and have multiple managers. For SAP BW, this employee will be included in the count of all employees only once, and not be included in every organization.



In addition to hierarchies in dimensions, there are hierarchies in SAP BW metadata (p. 193).

Levels for a Regular Dimension

A level is a collection of attributes, typically of a common granularity. Each level needs an item that is defined as a key and another item that is defined as a caption. For SAP BW data sources, levels contain members.

The first level of the hierarchy is automatically defined as the All level. It contains a single root member, which represents the top level of the hierarchy. For example, the All level for the Time dimension is named Time (All). You cannot delete or move the All level. You can change its name, description, and screen tip.

If you do not specify the levels of the hierarchy correctly, incorrect aggregation could occur.

Member Unique Names

The member unique name (MUN) is how the member is found in the data source, much like using business keys to find records in a table.

The member unique name is used in the expression for a member data item that is used in a report, a reference to members in filters and expressions, and used in drill-through between OLAP data sources. The member keys in the MUN for the different OLAP data sources must match.

Roles

Information about roles for relational metadata appears in a different topic (p. 123).

Roles define what appears in the member tree in the Cognos 8 studios. Use roles to organize and manage metadata and to determine how to present data to your users.

You can also create expressions that refer to roles instead of query items. You must use the `roleValue` function to refer to a particular role. For example, you want to query against a specific role in a hierarchy but the query item playing that role is different at each level of the hierarchy. A single query can span the different query items at each level. You can also use the `roleValue` function when you know the role but not the underlying query item.

You can assign multiple roles to one query item, but the same role cannot be assigned to different query items in the same level.

Default roles are pre-defined for all parent-child hierarchies and for all levels in level-based hierarchies. Most of these roles are not visible in the Cognos 8 studios.

The roles that are reserved by Cognos 8 start with an underscore. The name for a custom role cannot start with an underscore.

Default Roles

The default roles include the following:

- _businessKey

  Represents the key for the level. This role is also used to drill through from one data source to another because the business key should be consistent across your organization.

  The _businessKey role can be assigned to only one attribute in a level.

- _dimensionUniqueName

  Returns the name of the dimension as defined in the Framework Manager model.

- _hierarchyUniqueName

  Returns the name of the hierarchy as defined in the Framework Manager model.

- _levelLabel

  Returns the label that is assigned to the level.

- _levelNumber

  Returns the number for the level.

- _levelUniqueName

  Returns the name that is assigned to the level.

- _longName

  Is assigned to the query item that represents the long name for a level.

- _memberCaption

  Presents the caption for a member that will be shown in the Cognos 8 studios.

- _memberDescription

  Returns the description for a member within a dimension.

- _memberUniqueName

  Returns the Cognos member unique name.

- _parentUniqueName

  Defines the name that is assigned to the parent of the selected query item.

- _planningDefault

  Specifies which query item to use when measures are selected. This role is applicable only for Cognos Planning and SAP BW.

- _rollupType

  Defines how a query item is aggregated.

- _shortName

  Is assigned to the query item that represents the short name for a level.

If a query item uses a default role, you cannot change its role. This applies to SAP BW metadata only.

Custom Roles

You can create custom roles. Each role that you create must have a unique name. The roles that are reserved by Cognos 8 start with an underscore. The name for a custom role cannot start with an underscore. You can translate the custom roles in the model.

User-defined properties in OLAP data sources are assigned roles with the same name as the query item.

Specify Roles

Roles define what appears in the member tree in the Cognos 8 studios. Use roles to organize and manage metadata and to determine how to present data to your users.

Steps

1. Click the dimension whose roles you want to define.
2. From the **Actions** menu, click **Edit Definition**.
3. Click the **Dimension** tab.
4. In the **Hierarchies** box, click the level you want.
5. In the **Select a level in the hierarchy control to see the query items** box, click a query item.
6. Under **Role**, click the ellipsis (...) button.
7. Click the **Custom Roles** tab, and click **Add**.
8. Click **Close**.
9. Click **OK**.

You can also use the **Dimension Map** tab to define roles. Click **Attributes**, right-click the query item, and click **Edit Roles**.

Modify a Key Figures Dimension

Information about creating a measure dimension for relational metadata appears in a different topic (p. 124).

A key figures dimension is a collection of facts such as Quantity Sold or Price. Key figures are related to each other through the regular dimensions. When used in a report or analysis, the key figures dimension shows the value of the query item such as a name or number, or shows null, zero, or invalid. To create reports that fully compare and contrast functional areas, you may need to use more than one key figures dimension in a report.

You can add value by embedding calculations based on existing business rules, such as Profit Margin.

You can also modify the properties for multiple dimensions at the same time (p. 40).

You cannot define hierarchies or levels for a key figures dimension.

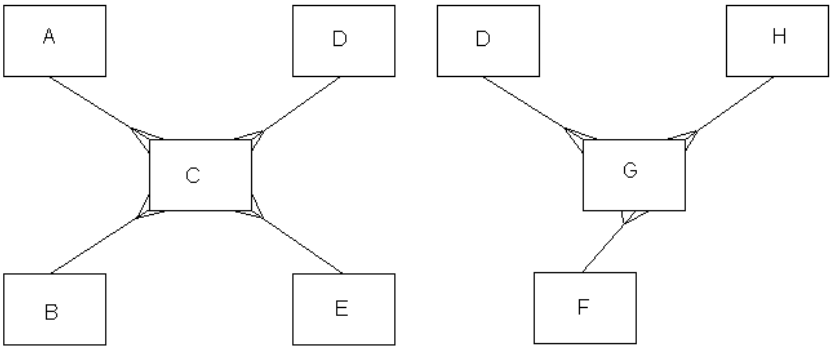
When dimensions are based on SAP BW metadata, you cannot edit the underlying query. However, you can add calculations and filters.

Steps

1. Click the key figures dimension you want to modify.
2. From the **Actions** menu, click **Edit Definition**.
3. Choose the action that you want:
   - Embed calculations by clicking **Add** and then defining the expression (p. 231).
   - Change the order of measures, items, or calculations by using the arrow buttons. However, if the measure dimension contains a folder, you can change the order only in the **Project Viewer**.
   - Embed filters (p. 233).
   - Test the dimension (p. 208).
4. Click **OK**.

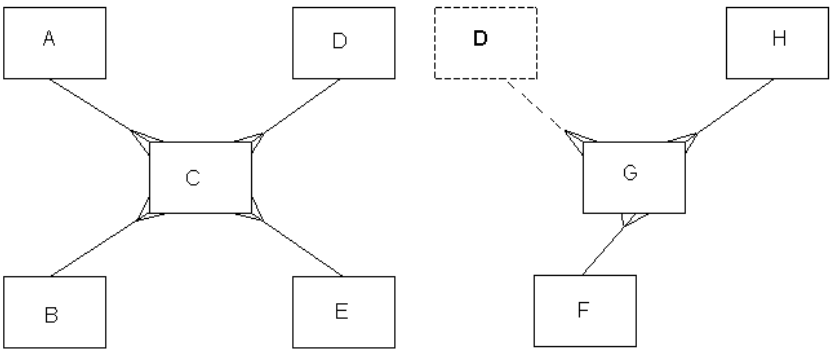
Conform SAP BW Dimensions

When you import multiple SAP BW Queries or InfoCubes, there can be two or more hierarchies that contain the same information.



You can conform these hierarchies in Framework Manager to create one dimension and one or more shortcuts. These hierarchies must have the same levels and properties. However, the order of the properties can be different. By conforming dimensions, your users can create reports that use multiple cubes without using an SAP BW MultiProvider.

A conformed dimension is represented by a dimension with one or more shortcuts to it. The dimension contains a list of data sources in which the dimension exists. If you want to retrieve data from a specific data source, ensure that you import the dimension from this data source before all others.



The dotted line in the diagram indicates a shortcut.

For example, a project contains three SAP BW Queries. One SAP BW Query contains basic information with dimensions and key figures, one contains the same information with variables applied, and one contains the same information with currency conversions applied. The common dimensions can be conformed allowing the modeler to maintain one set of common dimensions rather than three sets. To simplify your model, create a business view that uses one set of conformed dimensions, and create individual Key Figure dimensions to reflect the three different SAP BW Queries: Key Figures, Key Figures with Variables, and Key Figures with Currency Conversion.

When conforming dimensions, consider the scope of object-based security (p. 255).

Note that you cannot conform time or date dimensions. This is because values in the dimensions are often not the same across cubes, and this can produce errors and invalid reports.

Steps

1. Select one or more dimensions that you want to conform.
2. From the **Tools** menu, click **Generate Conformed Dimensions**.

The **Generate Conformed Dimensions** wizard appears, listing all dimensionally aware SAP BW dimensions that exist in the model.

3. Select the appropriate areas containing the dimension that you want to conform with, or select the actual dimension.

   If you want queries based on these dimensions to retrieve data from a specific data source, select the dimension that references the preferred data source.

4. Click **Next**.

5. Identify the dimensions that you want to include in the conformed dimension, and click **Next**.

6. Click **Finish**.

Remove a Conformed SAP BW Dimension

When you import SAP BW metadata into your project, you can select an option for Framework Manager to automatically create conformed dimensions for any two or more hierarchies that have a matching structure. If there are dimensions that you do not want conformed, you can remove them after the initial import.

Steps

1. Delete all instances of the dimension from the model.

   Tip: The easiest way to do this is to delete the one instance of the dimension that is not a shortcut. This removes all the shortcuts from the project.

2. From each relevant data source, re-import the dimensions that were deleted, ensuring that the **generate conformed dimensions** option is not selected.

3. Ensure that dimensions appear in the project as expected. No shortcuts should be used for these dimensions.

View Related Objects

Information about viewing related objects for relational metadata appears in a different topic (p. 127).

You can explore a visual representation of the objects that are connected to the query subject or dimension that you select in the **Project Viewer**. The **Context Explorer** shows the objects that the selected object is connected to. You can also select a connected object and see its references.

You can hide an object in the **Context Explorer**. You can also change the layout, fit all objects in the **Context Explorer**, and zoom in and out.

You can also use the **Dimension Map** tab to explore dimensions.

Steps

1. Select one or more objects that you want to explore.

2. From the **Tools** menu, click **Launch Context Explorer**.

3. To see the connected objects, click one or more objects and click the appropriate button.

Goal	Button
View the objects that are related to the selected object.	
View the immediate references for the objects.	
View all references for the objects.	

4. If you want to see details about an object, such as its relationships and query items, right-click the object, click **Diagram Settings**, and then select the details you want.

Test a Dimension or Other Object

Information about testing relational metadata appears in a different topic (p. 128).

You can test a dimension, a level, a hierarchy, query items, or model query subject.

Testing a regular dimension returns the attributes associated with the first hierarchy encountered in the dimension.

You can see the results that an object returns by testing it. You can test when creating an object or later on. The objects you can test are dimensions, query subjects, query sets, hierarchies, levels, calculations, and query items.

You can view the data that will appear in a specific report before publishing a package by selecting and testing the objects that will appear in the report. This makes it easier to debug a model and to verify that the model meets the reporting requirements because you do not need to create and publish packages first.

When you test an object, Framework Manager returns sample data. Formatting is not applied to the sample data. If you must test formatting, you must publish the package and view the objects in the Cognos 8 studios.

You may see different results depending on what you test. For example, if you use the expression editor to test a calculation that is embedded in a query subject, Framework Manager tests only the expression, not the item, so the aggregation setting for the query item is not applied to the test. Testing the entire query subject, which includes the calculation, gives a different result because the aggregation setting is applied. For example, if the aggregation setting is summarize, you can see a smaller number of rows in the test.

If you test a child segment of a segmented model, you may see an error if an object you are testing refers to an object in another child segment and the referenced object is not available to the project you are in. We recommend that you check that the parent model contains all the objects and that this error message does not appear when you test the parent model.

Governor settings may affect the testing results. For more information, see "Set Governors" (p. 310).

You can change existing test settings to customize the results that the test shows. For example, in addition to other settings, you can control the number of rows returned.

Steps When Creating or Modifying the Object

1. Select the object you want to test.
2. From the **Actions** menu, click **Edit Definition**, and click the **Test** or **Query Information** tab.

   The **Test Results** box is initially empty until you run the query.

   Any result sets that contain binary large objects are shown as [blob].
3. To run the query and bring back all the test results, click **Test Sample**.
4. If you are testing an expression and you want to apply the **Regular Aggregate** property of the query item or measure that is referenced in the expression, select the **Auto Sum** check box.

   If you clear this check box, a row is returned for each row in the result set of the query.
5. If you want to obtain more information about the query results, click the **Query Information** tab.
6. Click **OK**.

Steps to View the Data That Will Appear in a Specific Report

1. Select the objects that will appear in the report.
2. From the **Tools** menu, click **Test**.

   A message appears, telling you if the test was successful or if problems were found.
3. To view details about any problem that is found, click the **Query Information** tab.

   If you do not see the results of the query in the test window, the data from your data source may exceed the value of the **Runtime Limits** governor. The query stops at the specified limit, but the test result window does not contain any data. **Tip:** Set the **Runtime Limits** governor to zero.

Change the Test Settings

You can customize the tests by changing the test settings.

Steps

1. Select the object that you want.
2. From the **Actions** menu, click **Edit Definition** and then click the **Test** tab or the **Query Information** tab.
3. Click **Options** and then click the **Test Settings** tab.
4. Choose the options that you want.

Goal	Action	Persistence
Retrieve all data and show a specified number of rows	Select the **Restrict the maximum number of rows to be returned** check box and type the required number of rows. This setting does not improve performance for retrieving data when testing dimensions, query subjects, and query sets.	This setting applies to all dimensions, query subjects, and query sets in the model. This setting is saved and used in your next session with any model.
Specify the level of detail	Drag the **Level of Information shown in Results Information** slider to the location that represents the amount of detail you require.	This setting is saved and used in your next session with this model.
Temporarily override session parameters	In the **Session Parameters** box, click **Set**. The **Session Parameters** dialog box appears.	The override values are not saved with the model. This setting is for your current session only.
Apply relevant design mode filters	Select the **Apply all relevant design mode filters when testing** check box. This applies all relevant filters whose usage is set to design mode in another dimension, query subject, or query set.	This setting is saved and used in your next session with any model.
Apply a security filter	In the **Security Filters** box, click **Edit**.	This setting is saved and used in your next session with this model.
Change the prompt values	In The **Current Prompt Values** box, click **Prompts**. The **Model Prompts Manager** dialog box appears, which shows all prompts, and their values, that are in the model.	The prompt values are not saved with the model. This setting is for your current session only.

5. Click **OK** two times.

You may be interested in the following related topics:

- working with dimensions (p. 198)

- working with query subjects (p. 211)

Working with Model Query Subjects

Information about query subjects based on relational metadata appears in a different topic (p. 87).

A query subject is a set of query items that have an inherent relationship.

You use Framework Manager to modify query subjects to optimize and customize the data that they retrieve. For example, you can add filters or calculations. When you change the definition of a query subject, Framework Manager regenerates the associated query items, ensuring that any changes to query subject properties are reflected in all query items for that query subject.

For SAP BW metadata, you can work with model query subjects in Framework Manager.

Model query subjects are not generated directly from a data source but are based on query items in other query subjects or dimensions, including other model query subjects. By using model query subjects, you can create a more abstract, business-oriented view of a data source.

If you create a model query subject containing multiple dimensions, you may encounter problems when using the model query subject in conjunction with other query subjects or dimensions. Ensure that the items in a model query subject do not contravene the logic of the model, for example, the product item inserted between the country and city items. Test the model query subject in a report; if grouping works, the model query subject is valid.

Create a Model Query Subject

Information about query subjects based on relational metadata appears in a different topic (p. 87).

For SAP BW metadata, you can create only a model query subject.

You can also create a new model query subject by merging existing query subjects and query items (p. 212).

Steps

1. Select the namespace folder and, from the **Actions** menu, click **Create**, **Query Subject**.
2. In the **Name** box, type a name for the new query subject.
3. Click the **Model Query Subjects and Query Items** button, and click **OK**.

   The **Query Subject Definition** dialog box appears.
4. Click the **Query Subject Definition** tab.
5. From the **Available Model Objects** box, drag the items for this model query subject to the **Query Items and Calculations** box.

   You can change the order of items and calculations. However, if the query subject contains a query item folder, you can change the order only in the **Project Viewer**.
6. To embed calculations in the model query subject, click **Add** and define the calculation.
7. To embed filters in the model query subject, click the **Filters** tab.

8. To test the model query subject, click the **Test** tab.
9. Click **OK**.
10. To use a star layout in the Diagram view, from the **Diagram** menu, click **Auto Layout Diagram**, and then select **Star**.

Tip: To control the model area that is visible in the diagram, click and hold the overview button in the bottom right corner and drag the pointer over the diagram .

You may be interested in the following related topics:

- embedded calculations (p. 231)
- embedded filters (p. 233)
- testing and setting test options (p. 208)
- modifying the properties for multiple query subjects at the same time (p. 40)

Create a Model Query Subject Based on Existing Objects

Information about model query subjects based on relational metadata appears in a different topic (p. 98).

You can select existing model objects and merge them into a new model query subject. This means that you can reuse existing metadata to quickly create query subjects.

The objects that you can merge include

- model query subjects and their shortcuts
- query items, filters, and calculations in model query subjects

You can merge any number of the same type of objects into a new query in a single operation. The merge always creates a new model query subject.

The new query subject contains any filters that exist in the original query subject.

Notes:

- Ensure that model query subjects do not contravene the logic of the model. For example, if a query subject with multiple characteristics is used in combination with other query subjects, there can be problems when you run the report.
- Do not include query items from different query subjects or hierarchies from the same dimension. This causes a run-time error.

Steps

1. Ctrl+click the objects that you want to merge into a single query subject.
2. From the **Actions** menu, click **Merge in New Query Subject**.

Validate a Model Query Subject

Information about validating relational query subjects appears in a different topic (p. 106).

You can validate the definition of the query subject without having to open the **Query Subject Definition** dialog box. This is useful to do when

- new query items were added to a query subject
- the definition of the underlying query subject has changed

The **Evaluate Object** command completes an exhaustive check of all query subjects and ensures that they can run.

When Framework Manager evaluates a query subject, a request is sent to the SAP BW data source. Physical attributes, such as data type, are then updated as needed for the query subject.

You can also synchronize the entire project (p. 306).

Steps

1. Select the query subject you want to evaluate.
2. From the **Tools** menu, click **Evaluate Object**.

If you changed the **Regular Aggregate** property to **unsupported**, the property is reset when you evaluate the query subject. If the property is set to any other value, the property is not changed.

Query Items

Information about query items for relational metadata appears in a different topic (p. 139).

A query item is the smallest piece of the model that can be placed in a report. It represents a single instance of something, such as the date that a product was introduced.

Presentation hierarchy levels, key figures, and attributes are imported as query items in Framework Manager.

Only one hierarchy from a dimension should be used in the same report.

For SAP BW metadata, you can modify only text-based properties, such as the name or screen tip.

Because reports can contain different query items from one or more objects in the model, query item properties control many aspects of the final report. When you create a model dimension or model query subject, the query items inherit the properties of the data source query items on which they are based.

The properties for query items or measures include the following.

Query item property	Description
Name	The name of the query item or measure.
Description	A description of the query item or measure.

Query item property	Description
Last Changed	The date that the query item or measure was last changed.
Screen Tip	A description that can appear in the published package for your users.
Expression	Used to create embedded calculations that provide your users with calculated values that they regularly use. This property is for measures only. **Note:** The **Expression** property is not used by SAP BW.
External Name	The name that appears in the data source.
Is Hidden	Whether to hide or show the query item or measure in the published package. Even when **Is Hidden** is set to **True** and the query item or measure is invisible to your users, it is always present in the published package because the query item or measure may be needed by other objects in the model. You do not see the query item or measure in the **Package Publish** wizard. For example, a calculation may make use of a hidden query item.
Usage	The intended use for the data represented by the query item. This property is for query items only.
Format	How information appears in a report.
Currency	Which currency is used. This property cannot be changed in the **Property** pane. Use the **Format** property to change the currency.
Data Type	The data type that was set in the data source. Because this property is set in the data source, it is read-only in Framework Manager.
Precision	The total number of digits. Because this property is set in the data source, it is read-only in Framework Manager.

Query item property	Description
Scale	How many digits are represented in the scale. For example, you can show numbers in thousands so that 100,000 means 100,000,000. Because this property is set in the data source, it is read-only in Framework Manager.
Size	The size of the query item or measure. Because this property is set in the data source, it is read-only in Framework Manager.
Is Nullable	Whether the query item or measure can contain a null value. Because this property is set in the data source, it is read-only in Framework Manager.
Display Type	How the query item is shown. The column value can appear in the Cognos 8 studios as a picture, as a link, or as a value. The default is value. This property is for query items only.
MIME Type	The format that the column value uses. For example, if **Display Type** is set to picture, **MIME Type** could be jpeg. This property is for query items only. **Note:** The **MIME Type** property is not used by SAP BW.
Prompt Info	Prompt behavior.
Regular Aggregate	The type of aggregation that is associated with the query item, measure, or calculation in the published package.
Aggregate Rules	For dimensionally modeled relational metadata, the rules for semi-additive aggregation. For SAP BW metadata, the **Aggregate Rules** property is read-only.
Is Unsortable	Whether the values of this query item can be sorted. This property is for query items that contain large objects such as BLOBs.

You may be interested in the following related topics:

- the **Usage** and **Regular Aggregate** properties (p. 216)
- changing the currency symbol (p. 219)
- prompts (p. 220)
- modifying the properties for multiple query items at the same time (p. 40)

Modifying How Query Items Are Aggregated

Information about aggregation of relational query items appears in a different topic (p. 141).

You can change how some query items and measures are aggregated in reports. Framework Manager applies aggregate rules when your users create a report that summarizes a query item or measure.

When you import metadata, Framework Manager assigns values to the **Usage** and **Regular Aggregate** properties for query items and measures depending on the type of object that the query item or measure is in. The **Usage** property identifies the intended use for the data represented by the query item (p. 145). The **Regular Aggregate** property identifies the type of aggregation that is applied to the query item or measure (p. 146). Your users can override the values of the **Regular Aggregate** property. For semi-additive measures, you can specify additional aggregate rules by modifying the **Aggregate Rules** property (p. 146).

When modifying the **Regular Aggregate** property, you can select values that are not available through importing, such as average and maximum. You must understand what the data represents to know which aggregate rule is required. For example, if you aggregate a part number, the only aggregate values that apply are count, count distinct, maximum, and minimum.

Rules for Setting Properties for Dimensions

Framework Manager uses the following rules to set the **Usage** and **Regular Aggregate** properties.

Object	Usage property	Regular Aggregate property
Query item in a regular dimension	Attribute	Unsupported
Query item in a measure dimension	Identifier	Count
Measure in a measure dimension	Fact	Automatic if the measure is a calculation Sum if the measure is not a calculation

If the measure is semi-additive, use the **Aggregate Rules** property to define rules for semi-additive aggregation (p. 146).

For SAP BW metadata, you cannot change these properties for dimensions.

Rules for Setting Properties for Calculations

The **Regular Aggregate** property for a calculation in SAP BW metadata is set to **Automatic**. To determine what automatic means, these rules apply.

Calculation	Aggregation type
key items	unsupported
all other items	calculated

Rules for Setting Properties for Model Query Subjects

For model query subjects, Framework Manager uses the settings for the object that the model query subject is based on.

Note: If you change an aggregation value for SAP BW metadata, the aggregation cannot perform time-based queries because the aggregation rules are not applied.

Usage Property

The **Usage** property identifies the intended use for the data represented by each query item. During importing, the **Usage** property is set according to the type of data that the query items represent in the data source.

You need to verify that this property is set correctly. For example, if you import a numeric column that participates in a relationship, the **Usage** property is set to **identifier**. You can change the property.

For SAP BW query items, the value of the **Usage** property depends on the type of dimensional item the query item is based on.

Usage property value	SAP BW object	Description
Identifier	hierarchy level	Uniquely identifies characteristic values at a particular level in a hierarchy.
Fact	key figure	Represents a key figure that typically is numeric data. Date and time data are also supported.
Attribute	display attribute	Represents a display attribute that is associated with a characteristic.

Regular Aggregate Property

The **Regular Aggregate** property identifies the type of aggregation for the query item or calculation when you publish it. Your users can either use this default setting to perform calculations on groups of data, or apply a different type of aggregation.

For example, if the **Regular Aggregate** property value of the Quantity query item is sum, and it is grouped by Product Name in a report, the Quantity column in the report shows the total quantity of each product.

The following aggregation types are supported for SAP BW data sources:

- automatic
- average
- average non-zero is supported only when it is set in the data source. You cannot change the property to average non-zero in Framework Manager.
- calculated
- count
- count distinct
- count non-zero is supported only when it is set in the data source. You cannot change the property to count non-zero in Framework Manager.
- maximum
- median
- minimum
- standard deviation
- sum
- variance

Rules to Determine the Automatic Aggregation Type

If the calculation is in an SAP BW object, these rules apply.

Calculation	Aggregation type
key items	unsupported
all other items	calculated

Aggregate Rules Property

For dimensionally modeled relational metadata, the method by which a semi-additive measure is aggregated for the dimensions that you select.

For SAP BW metadata, the **Aggregation Rules** property is read-only. The **Semi-Aggregate** property is used instead.

Semi-Aggregate Property

For SAP BW metadata, the **Semi-Aggregate** property shows the value that is set in the data source, and the property is read-only.

If the value is set to **unsupported** in Framework Manager, the semi-aggregate behavior is ignored in the Cognos 8 studios.

The **Semi-Aggregate** property will not be supported in future releases. Instead, use the **Aggregate Rules** property for semi-additive measures.

Format Query Items

Information about formatting relational items appears in a different topic (p. 148).

You can specify how query item values appear in reports. Use the **Format** property to choose a format type, such as text, date, and currency. Each format type contains properties that further specify how the data appears.

For example, you can assign the **Currency** format type to a numeric query item, and then use the **No. of Decimal Places** property in the **Data Format** dialog box to specify how many decimal places appear in reports.

Some characters are language-sensitive and appear properly only when your locale supports the applicable font. For example, for Japanese currency symbols to appear correctly, your locale must be set to Japanese.

If Framework Manager does not show the currency you require, you must ensure that you install the appropriate language packs to support the currency symbols. For example, to have the Indian currency symbol (rupee) appear, you must run an operating system or install a language pack that can show this symbol. The Japanese operating system or Japanese language is one that can show the Indian currency symbol.

You can define properties for several query items at the same time. However, if the query items have different format types, all properties that were previously specified are overridden and the default values from the data source are used. If the original format types of the selected query items are the same, all the properties for the selected query items are set identically.

For example, to use the same decimal separator for two query items and to keep the number of decimals different, each query item must be changed individually. If both are selected and changed at the same time, all properties including the number of decimals are set identically for both query items.

Steps

1. In the **Project Viewer** pane, click the query item you want to format.
2. In the **Properties** tab of the **Properties** pane, click the **Format** property.

3. Set the format type to currency to ensure that currency formatting is applied to all types of reports.
4. In the **Currency scope** box, specify the type of currency. If you do not see the currency you want to use, click the **Add** button.
5. In the **Properties** box, select or type the appropriate property value.
6. Click **OK**.

Define a Prompt Control

Information about prompt controls for relational metadata appears in a different topic(p. 149).

Prompts help your users quickly find the information they need in a report. Prompts are generally defined in reports. However, you can change the definition of the dimension or query subject in the model so that a prompt appears automatically when your users create filters. This is useful for query items, such as ProductTypeCode, whose values are not shown in a report but are useful for filtering data.

In general, it is better to define type-in prompts in the reports to make use of the additional prompt features. However, your users cannot modify some variables. For these variables, you can use Framework Manager to define type-in prompts.

The syntax for using a prompt as a value is
`?<PromptName>?`

You can use prompts in

- parameter maps
- session parameters
- expressions, including filters and calculations

Steps

1. Click the query item.
2. In the **Properties** pane, click the **Properties** tab.
3. Click the plus sign (**+**) next to the **Prompt Info** property.

   This is a compound query item property.
4. Modify the following properties to reflect the behavior you require.

Goal	Property
Set the type of prompt control that is generated when the report is run.	Prompt Type

Goal	Property
Set the generated prompt as part of a series of generated cascading prompts.	Cascade On Item Reference
Specify that a prompt defined for one query item will use a different query item, and its rules, in the prompt. The query item specified in the **Display Item Reference** property will display a list of values in the prompt. Each value in the list will be associated with a value in the query item specified in the **Use Item Reference** property.	Display Item Reference
Specify that a prompt defined for one query item will use a different query item, and its rules, in the prompt. The query item specified in the **Use Item Reference** property will be used in the prompt.	Use Item Reference
Set an alternative query item that is used by any Report Studio or Query Studio filter.	Filter Item Reference

Prompt Type Property

The **Prompt Type** property sets the type of prompt control that is generated when the report is run, such as an edit box or a pull-down list. Use this property to have the prompt show one query item but use a different one. For example, the prompt can show Employee Name but use Employee Number.

The default value for this property is **Server Determined**.

Value	Prompt Control
Server Determined	The type of prompt control is based on information in the server, such as the data type of the query item.
Edit Box	A simple text box. If the data type of the column is date or dateTime, this value generates a date or date-time control as well as the text box.
Select Date	A date control with a calendar interface.
Select Date/Time	A date-time control with a calendar interface. For SAP BW metadata, this value is not relevant.

Value	Prompt Control
Select Interval	A date-time interval control. For SAP BW metadata, this value is not relevant.
Select Time	A time control that filters data based on the selected time period. For example, if you define a Select Time prompt for Order Time, the user can use the time control to show all orders placed after 1:00, or all the orders placed between 10:00 and 11:00. If you are referring to a time member, you must use the exact values only. If you are using a range, the end points of the range must correspond to values in the data source.
Select Value	A drop-down list.
Select with Search	A list control so that users can search for values. For SAP BW metadata, this value is not relevant.
Select with Tree	A tree prompt control for prompts that are based on a hierarchy node.

Cascade On Item Reference Property

The **Cascade On Item Reference** property indicates that the generated prompt is part of a series of generated cascading prompts. The query item that you reference in this property is the parent item in the cascade. The system prompts the user for the cascade item before prompting them for the current query item.

For example, if you want to prompt for Product Line and then Product within the selected line, set the **Cascade On Item Reference** property of the Product query item to Product Line.

Display Item Reference and Use Item Reference Properties

These properties specify that a prompt defined for one query item will use a different query item, and its rules, in the prompt.

The query item specified in the **Display Item Reference** property will display a list of values in the prompt. Each value in the list will be associated with a value in the query item specified in the **Use Item Reference** property.

The query item specified in the **Use Item Reference** property will be used in the prompt.

For example, you want the prompt to display Country Name while using Country Code to retrieve data. Set the **Display Item Reference** property to Country Name and set the **Use Item Reference** property to Country Code.

These properties are used only for data driven prompt controls whose **Prompt Type** property is set to either **Select Value** or **Select with Search**.

Filter Item Reference Property

The **Filter Item Reference** property identifies an alternative query item that is used by any Report Studio or Query Studio filter. This property is used only when Report Studio or Query Studio generates a filter. This property helps create more efficient queries because it is faster to filter on an indexed numeric column than a non-indexed string column.

For example, a report author wants to create a filter for the Country query item. You set the **Filter Item Reference** property to use Country Code for any filter that uses the Country query item.

Another example is a report author wants to create a filter for the Country Code query item that appears in the Orders table. You want that filter to use the Country Code in the Country table because there are fewer rows to read in the Country table.

Testing a Prompt

When you test a model object that references a prompt, Framework Manager asks you to enter the prompt value. Framework Manager uses this value for either the duration of the session, or until you clear the prompt value.

You can change the session value of prompt values through the **Options** dialog box. This dialog box is available when you modify a dimension or query subject, or define a calculation, filter, query set, or complex relationship. You can change the prompt value at the time that you are testing the expression that references that value.

If you select the **Always prompt for values when testing** check box in the **Prompt** dialog box, Framework Manager prompts you for a value every time you test the object. When updating the object or performing a count, Framework Manager uses the existing prompt value, if one exists.

A prompt on a query item in a model query subject is associated only with that query item. A prompt on a query item in a data source query subject is associated with the entire query subject and therefore, the prompt appears when you test any query item in the query subject.

SAP BW Variables

SAP BW variables are parameters of an SAP BW Query that are set up during query definition. When you run the query, the SAP BW variables are filled with values. They function as placeholders and can be processed in different ways. They are automatically exposed as prompts at run time.

SAP BW variable information is included in a composite custom property named **SAP BW Variables** that exists only if a data source has one or more variables associated with it. The **SAP BW Variables** property contains one or more composite properties, each of which must be assigned a unique name. Each property represents a description of a single SAP BW variable. Because the variable information is specified in a custom property, Framework Manager does not validate these properties.

Framework Manager supports these types of SAP BW variables:

- characteristic

There are two kinds of characteristic variables: characteristic value and hierarchy node. Characteristic values variables select characteristic values. Hierarchy node variables select values from any position in a presentation hierarchy.

- hierarchy

  The user is not prompted for a value because Cognos 8 automatically populates it at run time based on the selected hierarchy.

- formula

  The user types a numeric value at run time. Use formula variables if a formula component should be entered only when a query is run. For example, you can use a formula variable for a value-added tax rate to process the current rate at run time.

- authorization

  Authorization variables are like other variables, but Cognos 8 automatically populates the variable values with the user's credentials. SAP BW uses these credentials to supply the information needed by an SAP BW Query that has security applied to it.

Variables for hierarchies function as placeholders for the hierarchy of a characteristic. All the values for hierarchy variables are read-only.

The SAP BW variable information is obtained using the SAP BW BAPI *MDDataProviderBW::GetVariables*.

Name Property

This property is a string value.

SAP BW equivalent: VARIABLE_NAME

Restrictions: Read-only.

Caption Property

The string value for this property is a composite and locale-dependent. Each locale in the model should be represented by a custom property whose value is the locale name. For example, if the locales en-ca and fr-fr exist in the model, define two custom properties named en-ca and fr-fr.

The default value is obtained from SAP BW.

Default Low Caption and Default High Caption Properties

The value for each of these properties is a composite, locale-dependent string value. Each locale in the model should be represented by a custom property whose value is the locale name. For example, if the locales en-ca and fr-fr exist in the model, define two custom properties named en-ca and fr-fr.

The default value is obtained from SAP BW.

Restrictions: The **Default High Caption** property is applicable only for variables with a **Selection Type** of **interval**.

Fixed Value Property

This property is a boolean property that determines whether the value of the variable can be changed. If this property is set to **true**, your users are not prompted for the associated variable, and the default value is always applied.

The default value is **false**.

Restrictions: This property is not applicable to hierarchy variables.

Selection Type Property

The possible values are value, interval, complex, multiValued.

Value	SAP BW Equivalent
value	SAP_VAR_SEL_TYPE_VALUE
interval	SAP_VAR_SEL_TYPE_INTERVAL
complex	SAP_VAR_SEL_TYPE_COMPLEX
multiValued	SAP_VAR_SEL_TYPE_COMPLEX

Restrictions: Read-only.

Entry Type Property

The default value is obtained from SAP BW.

Value	SAP BW Equivalent
optional	SAP_VAR_INPUT_TYPE_OPTIONAL
mandatory	SAP_VAR_INPUT_TYPE_MANDATORY
mandatoryNotInitial	SAP_VAR_INPUT_TYPE_MANDATORY_NOT_INITIAL

Restrictions: Read-only.

Data Type

The default value is obtained from SAP BW.

Value	SAP BW Equivalent
xsdString	CHAR

Value	SAP BW Equivalent
xsdDate	CHAR The VAR_TYPE value is SAP_VAR_TYPE_MEMBER and the reference dimension is based on 0CALDAY.
xsdFloat	FLTP
xsdDatetime	DATS
xsdInt	NUMC, DEC, INT1, INT2, or INT4
xsdLong	NUMC, DEC, INT1, INT2, or INT4
xsdMUN	CHAR

Restrictions: Read-only.

Default Low Value and Default High Value Properties

Each of these properties specifies a range of values.

The default value is obtained from SAP BW.

Restrictions: The **Default High Value** property is applicable only for variables with a **Selection Type** of **interval**.

Description Property

This property is a string value.

SAP BW Variable Type Property

The possible values are numeric, characteristic, hierarchy, or hierarchicalNode.

The default is obtained from SAP BW.

Restrictions: Read-only.

Prompt Type Property

The default value depends on the type of the variable. If the value of this property is not one of the predefined values, it is assumed to be hierarchyPickList.

Value	Restrictions
typeIn	Required for numeric variables and optional for characteristic values
pickList	Optional for characteristic variables

Value	Restrictions
calendar	Only for characteristic variables based on 0CALDAY
hierarchyPickList	Optional for all presentation hierarchies
notApplicable	Required for hierarchy variables

Restrictions: Read-only for some types of variables such as characteristic and formula.

Level Restriction Property

This property is a numeric value.

The default value is **1**.

Restrictions: Applicable only for hierarchical node variables with a **Prompt Type** of **hierarchyPickList**.

Trim Levels Property

This property is a string value that reduces the number of members in a hierarchical picklist. If the property is set to zero (0), members from all levels of a hierarchy are included in the prompt. You can also specify a range such as 2:4 to include only the members from certain levels. If the starting and ending ranges are the same, such as 3:3, only members from that level will be included.

The default value is zero (0).

Restrictions: Applicable only for characteristic variables with a **Prompt Type** of **hierarchyPickList**.

Use Default Values Property

This property is a boolean property that determines whether the default values will be used. If this property is set to **true**, your users are not prompted for the associated variable, and the default value is always applied.

The default value is **false**.

Initial Number of Pick List Values Property

A numeric value that specifies the initial number of values used to populate a picklist, hierarchical picklist, or prompt.

The default value is zero (0), which means all.

Show Key and Caption Property

To show keys and captions for SAP BW variables, set this property to **true**. This property is applicable only for pick list prompts and hierarchy node prompts.

The default value is **false**.

Numeric Variable Property Values

The following variable properties are applicable to numeric variables:

Property	Default value
Type	numeric
Caption	
Selection Type	value
Entry Type	
Data Type	
Default Low	
Default High	
Prompt Type	typeIn
Fixed Value	false

You can change the default values for a numeric variable except for the **Prompt Type** property, which is read-only.

Hierarchy Variable Property Values

The following variable properties are applicable to hierarchy variables:

Property	Default value
Type	hierarchy
Caption	
Selection Type	value
Entry Type	obtained from SAP BW
Data Type	xsdString
Default Low	
Default High	

Property	Default value
Prompt Type	not applicable

Because hierarchy variables are resolved entirely at run time, all these properties are read-only.

Characteristic Variable Property Values

There are two kinds of characteristic variables: characteristic value and hierarchy node. Characteristic values variables select characteristic values. Hierarchy node variables select values from any position in a presentation hierarchy.

Characteristic Value Variable Property Values

The following variable properties are applicable to characteristic value variables:

Property	Default value
Type	characteristic
Caption	
Selection Type	obtained from SAP BW
Entry Type	obtained from SAP BW
Data Type	xsdString
Default Low	If the entry type is value or complex, the default property is shown. If the entry type is interval, the default low property is shown. This value is obtained from SAP BW.
Default High	If the entry type is value or complex, the default property is shown. If the entry type is interval, the default high property is shown. This value is obtained from SAP BW.
Prompt Type	typeIn or pickList This depends on the number of members in the referenced dimension. If the value is invalid, typeIn is used.
Fixed Value	false

A characteristic value variable for the 0CALDAY dimension is shown in the model as a date. The **Data Type** property is set to **xsdDate** and the **Prompt Type** property is set to **calendar**. The **Prompt Type** property is read-only for the 0CALDAY dimension.

Hierarchy Node Variable Property Values

The following variable properties are applicable to hierarchy node variables:

Property	Default value
Type	characteristic
Caption	
Selection Type	value
Entry Type	obtained from SAP BW
Data Type	xsdString
Dimension	obtained from SAP BW
Hierarchy	obtained from SAP BW
Default Low	
Default High	
Prompt Type	hierarchy PickList You can change the **Prompt Type** property to **typeIn** or **pickList**.
Trim Levels	zero (0)
Fixed Value	false
Maximum Number of Values	100

Picklist Prompts

Each picklist prompt contains a pre-defined number of values. These values are determined by the **Maximum Number of Values** property.

If the number of actual values is less than or equal to the default number of values, the prompt is generated as a single picklist prompt. If the number of actual values exceeds the default number, two prompts are generated in this order:

- a bound range parameter with a starting value of **1** and an ending value determined by the **Maximum Number of Values** property

  This parameter is of the type xsdUnsignedLong and is optional. The name of the parameter is the name of the original prompt followed by _range_prompt. The caption is locale-specific. If this is a multilingual model, you must store the template for the caption in a message file.

- a picklist prompt containing the default number of values

Adding Business Rules

Information about business rules for relational metadata appears in a different topic (p. 153).

Business rules that were created in SAP BW are imported into Framework Manager. You can add more business rules to your model to refine the retrieved data and to ensure that the right information is available for your users.

Creating business rules and storing them in the model instead of in reports has many advantages. You save time because you and your users do not have to re-create the business rules every time they are needed. The business rules ensure consistency because your users all use the same definitions. For example, Low Margin means the same thing throughout the organization. They are easy to update because you maintain the business rules centrally so that all reports are updated automatically as the rules evolve. For example, if the definition for Low Margin changes, all reports that use the Low Margin calculation are updated automatically. The business rules enhance security. For example, you can use filters to limit the data that your users can see.

You can

- add calculations so that your users can include calculated data in their reports (p. 231).
- create and apply filters so that you can limit the data that a query subject retrieves (p. 233).
- add prompts that will automatically appear whenever a dimension or query subject is used in a report; report consumers are then prompted to filter data (p. 220).
- use session parameters (p. 238) and parameter maps (p. 237) to dynamically resolve expressions.
- create a security filter to control the data that is shown to your users when they set up their reports (p. 253).

Create a Calculation

Information about calculations for relational metadata appears in a different topic (p. 153).

You can create calculations to provide your users with calculated values that they regularly use. Calculations can use query items, parameters, variables, calculated members, expressions, and expression components, such as functions.

Punctuation characters, such as the question mark (?), must be in 7-bit ASCII character code. If you type a punctuation character from a multi-byte enabled keyboard, ensure that you type the 7-bit ASCII representation of the character. For example, type Alt+063 for the question mark.

Avoid using characters that are used for expression operators in the name of the calculation. Syntax errors may occur when the expression is evaluated. For example, a calculation named Margin * 10 causes errors when used in an expression such as [Margin * 10]< 20.

At query time, Framework Manager returns a null value for any calculation that contains a divisor whose value is zero. Framework Manager cannot detect zero-division errors in functions such as average and mod, because the division operator is not explicit.

You can apply a stand-alone calculation to one or more dimensions or query subjects to provide calculated data to a report, or include it in a package to make it available to your users. By moving a stand-alone calculation or a shortcut to it into a folder, you can better organize the model objects.

Steps

1. Click the namespace or folder and, from the **Actions** menu, click **Create**, **Calculation**.
2. In the **Name** box, type a name for the calculation.
3. Define the expression.

Goal	Action
Add items	On the **Model** tab, click a query item, filter, or calculation and click the arrow.
Add functions	On the **Functions** tab, choose a component and click the arrow.
Add parameters	On the **Parameters** tab, click a parameter and click the arrow.
Retrieve all data and show a specified number of rows	Click the options button, select the **Restrict the maximum number of rows to be returned** check box, and type the required number of rows to be returned. This setting does not improve performance for retrieving data when testing dimensions and query subjects.
Override session parameters	Click the options button, click **Set**, enter a value in the **Override Value** field, and click **OK**.

Goal	Action
Override prompt values	Click the options button, and then click **Prompts**. The **Model Prompts Manager** dialog box appears, which shows all prompts, and their values, that are in the model.

4. To test the calculation, click the test button .

   You can test only calculations that contain query items. If a calculation contains a function, for example _add_days, the test sample button is not available.

5. Click **OK**.

6. Modify the **Data Type** property to identify the type of data the calculation returns.

   The Cognos 8 studio uses this information to format the data that the calculation returns.

For information about functions, see "Using the Expression Editor" (p. 387).

Create a Filter

Information about filters for relational metadata appears in a different topic (p. 156).

A filter is an expression that specifies the conditions that rows or instances must meet to be retrieved for the dimension, query subject, calculation, or report to which the filter is applied. A filter returns a boolean value so that you can limit the rows returned by a dimension or query subject.

For example, you can use the `in_range` function to create a filter that retrieves data for products introduced in a specific time frame. The syntax for this example looks like this:

```
[gosales_goretailers].[Products].[Introduction
date]
in_range {Feb 14, 1999 : July 14, 2007}
```

Note: When using a date or time function, you must use a 24-hour clock. Framework Manager does not support "a.m." or "p.m." in expressions. For example, use 20:00 to signify 8 p.m.

You can restrict the data represented by dimensions or query subjects in a project by creating a security filter. The security filter controls the data that your users can see when they set up their reports.

You can also apply governors to restrict the data that the queries in a package retrieve.

Framework Manager supports stand-alone filters and embedded filters.

- Use a stand-alone filter when you want to reuse the expression.

  You can add a stand-alone filter to one or more dimensions or query subjects to limit the data that the query retrieves when the filtered dimension or query subject is used in a report, or you can include it in a package to make it available to your users. By moving a stand-alone filter or a shortcut to it into a folder, you can better organize the model objects.

- Use an embedded filter when you want to use a filter with only one dimension or query subject.

  You can create an embedded filter when modifying a dimension, relational data source query subject, or model query subject.

  If you start with an embedded filter, you can later convert it into a stand-alone expression that you can apply to other dimensions or query subjects. **Tip:** Right-click the filter expression in the **Filters** tab and click **Convert to Stand-alone Filter**.

  When you embed a filter, the data source query subject must have a relationship to any query subject referenced by the expression. This relationship is necessary even if the expression references a model query subject based on the same table as the data source query subject in which you are embedding the expression.

  To create a filter on an unrelated query subject, do one of the following:

  - Ensure that there is a join path between the new query subject and the one that contains the filter.
  - Base the embedded filter on a query item that is based on the data source query subject you want.
  - Convert the calculation to a stand-alone filter, so that it is not part of the query subject.
  - Create a stand-alone filter that references the embedded object.

Steps

1. Do one of the following:
   - If you want to create a stand-alone filter, click the namespace or folder and, from the **Actions** menu, click **Create**, **Filter**.
   - If you want to create an embedded filter, double-click the dimension or query subject that will contain the filter, click the **Filters** tab, and then click **Add**.
2. In the **Name** box, type a name for the filter.
3. Define the expression.

Goal	Action
Add query items and filters	On the **Model** tab, drag the objects you want to the **Expression Definition** box.
Add functions	On the **Functions** tab, drag the functions to the **Expression Definition** box.
Add parameters	On the **Parameters** tab, drag the parameters to the **Expression Definition** box.

Goal	Action
Retrieve all data and show a specified number of rows	Click the options button, select the **Restrict the maximum number of rows to be returned** check box, and type the required number of rows to be returned. This setting does not improve performance for retrieving data when testing dimensions, query subjects, and query sets.
Override session parameters	Click the options button, click **Set**, enter a value in the **Override Value** field, and click **OK**.
Override prompt values	Click the options button, and then click **Prompts**. The **Model Prompts Manager** dialog box appears, which shows all prompts, and their values, that are in the model.

4. To test the filter, click the **test** button .
5. Click **OK**.

You may be interested in the following related topics:

- security filters (p. 253)
- functions "Using the Expression Editor" (p. 387)
- parameters (p. 237)
- session parameters (p. 238)

You can also apply governors to restrict the data that the queries in a package retrieve (p. 310).

Apply a Filter

Information about filters for relational metadata appears in a different topic (p. 158).

To apply a filter, you must modify the dimension, data source query subject, or model query subject. The query subject must either contain the query items that the filter references, or have a relationship path to the query subjects that contain the query items.

You can embed a stand-alone filter in dimensions or query subjects, but if you want a different usage for each embedded filter, you must create different versions of the stand-alone filter. Otherwise,

your users could be required to fill in a prompt that you thought was optional if there is any instance where the usage is set to mandatory.

For example, in query subject A, you embed a stand-alone filter and define it as optional. In query subject B, you define it as mandatory. When your users create a report that uses both query subjects, they are required to choose values in both filters, even the one defined as optional. All instances of the filter are considered to be mandatory when used in the same query. The solution is to create different versions of the filter, each with its own name.

Steps

1. Create a filter.
2. Select the filter and, from the **Actions** menu, click **Edit Definition**.
3. Click the **Filters** tab, and drag the filter you created to the **Filters** box.
4. Select a usage value for the filter.

Usage Value	Description
Always	Use this usage value to ensure specified data is filtered out of all reports. For example, your company may have obsolete information that it stores but does not want to report on. **Always** is the default usage value.
Design Mode Only	Retrieves a small subset of the data for the sample report. Use this usage value when you do not need to see all the data, for example when testing a query subject. Your users may need the design mode filter in Query Studio when they want to focus on designing the layout and format of a report and not retrieve all the data as they work. To access the design mode filter in Query Studio, run the report with limited data.
Optional	Specifies that the filter is optional. The user is prompted to filter data and can leave the prompt blank. If the prompt is blank, Framework Manager ignores the filter and retrieves all data for the dimension or query subject. The `? ?` syntax is required for optional prompts. Use this usage value if your users want to control when the filter is applied. For example, you want to see on one country sometimes and see the data for all countries at other times. An optional filter for country looks like this: `([GeoNamespace].[Countries].[CountryName] = ?WhichCountry?)`

5. If you want to view the SQL, click the **Query Information** tab.
6. Click **OK**.

Create a Parameter Map

Information about parameter maps for relational metadata appears in a different topic (p. 161).

Use parameters to create conditional query subjects that allow for substitutions when the report is run. Parameter maps are objects that store key-value pairs. Parameter maps are similar to data source look-up tables. Each parameter map has two columns, one for the key and one for the value that the key represents. You can manually enter the keys and values, import them from a file, or base them on existing query items in the model.

All parameter map keys must be unique so that Framework Manager can consistently retrieve the correct value. Do not place quotation marks around a parameter value. You can use quotation marks in the expression in which you use the parameter.

The value of a parameter can be another parameter. However, you must enclose the entire value in number signs (#). The limit when nesting parameters as values is five levels.

When you use a parameter map as an argument to a function, you must use a percentage sign (%) instead of a dollar sign ($).

We recommend that you assign an alias to a query item that uses a parameter map as part of its name and to add the multilingual names to the object in the **Language** tab (**Properties** pane).

Note: If you are using SAP BW metadata, you cannot use a query item to generate the keys and values of a parameter map.

Steps to Manually Create a Parameter Map

1. Click the **Parameter Maps** folder and, from the **Actions** menu, click **Create**, **Parameter Map**.
2. In the **Name** box, type a name for the new parameter map.
3. Click **Manually enter the parameter keys, and/or import them from a file** and click **Next**.
4. Do one of the following:
   - To manually enter values, click **New Key**, type a key, and press Tab to enter a value for that key.
   - To import keys and values, click **Import File** and identify the location of the appropriate .csv file. You can import from a .txt file only if the values are separated by tabs rather than commas.

   Note: If you are going to use a parameter in a data source query subject, the value must use English-specific punctuation. This means that you must use a period (.) to represent a decimal and a comma (,) to separate lists of values.
5. Modify existing parameters as required.

Goal	Action
Assign a default value	In the **Default Value** box, type a value. If the key is used in an expression that is not mapped, the default value is used.
Remove a parameter	Select a row and click **Delete**.
Modify a parameter	Select the row you want to modify, click the **Edit** button, and type a value.
Clear all keys and values	Click **Clear Map**.

6. Click **Finish**.

Default Values and Parameter Maps

The **Default Low Value** and **Default High Value** properties may contain expressions that use parameter maps. You can use parameter maps to define a value for a target currency variable based on the user's locale. For example, you define a parameter map that provides an ISO currency code. The value for the **Default Low Value** property could be defined as `#$Currency_Map[runLocale}#`

This parameter map is used when the SAP BW variable Target Currency is used in a report.

These are the only properties related to SAP BW variables that can use parameter maps.

Create a Session Parameter

Information about session parameters for relational metadata appears in a different topic (p. 163).

A session parameter is a variable that Framework Manager associates with a session. For example, user ID and preferred language are both session parameters. Because session parameters are key and value pairs, you can think of each session parameter as an entry in a parameter map named Session Parameters. You use a session parameter in the same way that you use a parameter map entry, although the syntax for session parameters is slightly different.

There are two types of session parameters: environment and model.

Environment session parameters are predefined and stored in Content Manager. By default, the following session parameters appear in Framework Manager:

- runLocale
- account.defaultName
- account.personalInfo.userName

If you log on anonymously, you will see only runLocale and account.defaultName(Anonymous).

If your authentication source supports other parameters and you entered information about them in the authentication source, you see other session parameters, such as account.personalInfo.email.

You can define additional parameters by using model session parameters. Model session parameters are stored in a parameter map named **_env**. They are set in the project and can be published with a package.

Model session parameters must have their values set within the scope of objects in the Framework Manager model. The scope can include the use of existing environment session parameters, as well as static values.

Each session parameter must have a name and a default value. You can define an override value to test the results that value returns. The override value is valid only when you have the model open, and is not saved when you save the model. If no override value exists, Framework Manager uses the default value when it executes a query that contains a session parameter.

The rules governing the use of parameters include the following:

- All possible return values must have the same data type.
- Only one value can be defined.

Steps

1. From the **Project** menu, click **Session Parameters**.
2. Click **New Key** and type a session parameter key and value.
3. Choose how to handle the override value.
   - To avoid having to set the override value every time you edit the project, set the session parameter as a value.
   - To avoid having to remove the project setting each time before you publish it, set the session parameter as a session override.
4. Modify existing parameters as required.

Goal	Action
Change the parameter value	Click the row that contains the value you want to change, click **Edit**, and type a value.
Assign a default value	In the **Default Value** box, type a value. Framework Manager uses the default value if a key has an invalid value.
Remove a parameter	Click a row and click the **Delete** button. You cannot delete an environment session parameter.
Clear an override value	Click a row and click **Clear Override**.

5. Click **Finish**.

Organizing the Model

Information about organizing a relational model appears in a different topic (p. 175).

When you organize the model, you make it easier for your users to find and understand the data in the model. You also make the model easier for you to manage and maintain. We recommend that you create several views, or layers, in the model:

- Keep the metadata from the data source in a separate namespace or folder.

  In Framework Manager, this is called the import view.

- Create one or more optional namespaces or folders for resolving complexities that affect querying using query subjects or dimensional objects.

  To use Analysis Studio or any OLAP-style queries, there must be a namespace or folder in the model that represents the metadata with dimensional objects.

- Create one or more namespaces or folders for the augmented business view of the metadata that contains shortcuts to dimensions or query subjects.

  In Framework Manager, these are called the business view. Use business concepts to model the business view. One model can contain many business views, each suited to a different user group. You publish the business views.

Security can be defined in any of the views. It depends on your business requirements. For example, if you need to keep everyone from viewing an object, you add security to the object in the import view. Typically security is applied in the business view.

You can

- include metadata in several folders by using shortcuts (p. 240)
- organize objects by creating namespaces or folders (p. 242)

Use Shortcuts

Information about shortcuts for relational metadata appears in a different topic (p. 180).

A shortcut is a pointer to an object, such as a relationship, a dimension, a query subject, or a folder. We recommend that you use shortcuts in the business view when there is an overlap between user groups and you want to include the metadata in more than one folder. With shortcuts, you can have multiple references to an object.

For example, you create folders named Orders, Products, and Customers. If you want both Orders and Customers to contain the same dimension, you must create a shortcut to the dimension and add it to both folders.

When you create a shortcut, Framework Manager does not set the **Screen Tip** and **Description** properties. Unless you define these properties, the values shown in the Cognos 8 studios are the same as those defined in the object that the shortcut references.

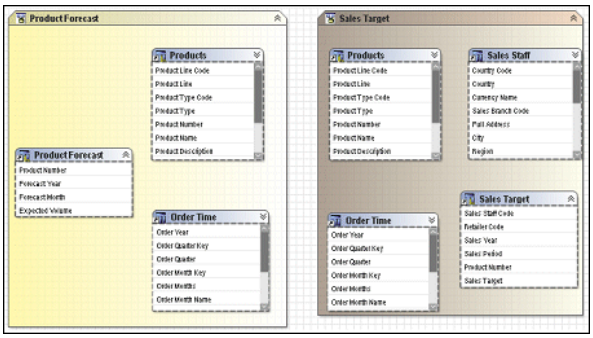
Tip: To go to the object that the shortcut references, right-click the shortcut and click **Go To Target**.

Shortcuts are less flexible from a presentation perspective than model objects, but they require much less maintenance because they are automatically updated when the target object is updated. If maintenance is a key concern and there is no need to customize the appearance of the query subject, use shortcuts.

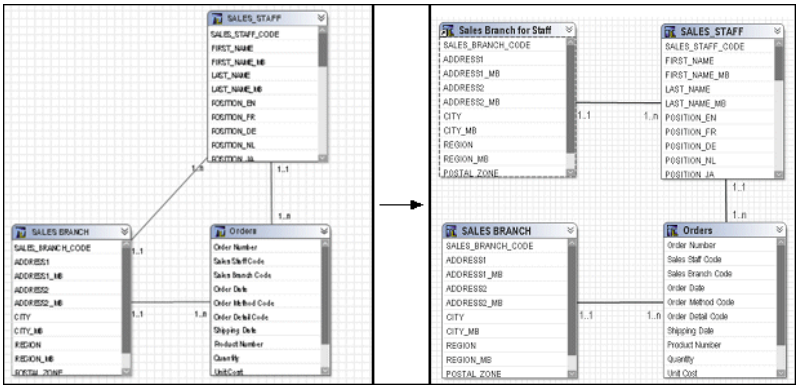
Framework Manager has two types of shortcuts:

- regular shortcuts, which are a simple reference to the target object.
- alias shortcuts, which behave as if they were a copy of the original object with completely independent behavior. Alias shortcuts are available only for query subjects and dimensions.

Regular shortcuts are typically used as conformed dimensions with star schema groups, creating multiple references with the exact same name and appearance in multiple places. In the example below, the shortcuts created for Products and Order Time behave as references. If a query is written that brings Products from both Product Forecast and Sales Target, the query uses the definition of Products based on the original and this definition appears only once in the query.



Alias shortcuts are typically used in role-playing dimensions or shared tables. Because there is already an example in this document for role-playing dimensions, we will look at the case of shared tables. In this example, Sales Staff and Sales Branch can be treated as different hierarchies. From our knowledge of the data, we know that because staff can move between branches, we need to be able to report orders against Sales Branch and Sales Staff independently as well as together. To achieve this, we need to create an alias to Sales Branch that can be used as a level in the Sales Staff hierarchy.



With the new alias shortcut in place, it is possible to create queries that require orders by sales branch and orders by sales staff with their current branch information simultaneously.

Shortcuts and Dimensions

Shortcuts result in fewer dimensions to maintain. You can keep dimensions in the import view and keep shortcuts in the business view.

When you create a shortcut to a dimension, you cannot customize which query items are in the shortcut. The entire dimension is included in the shortcut.

If you want to represent conformed dimensions in several star schema groups, only one dimension or query subject can exist for a dimension table. Use shortcuts for all other instances of the dimension or query subject. By using shortcuts, you can build queries involving multiple fact tables that are related through shared dimension tables.

The security you specify for an object is passed to shortcuts that reference the secured object. If you have a shortcut to a secured object, only users with permission to see the secured object can see the shortcut in the published package.

Step

- Right-click the query subjects, dimensions, or folders that you want to create shortcuts to, and click **Create Shortcut**.

Create a Folder or Namespace

Information about folders and namespaces for relational metadata appears in a different topic (p. 183).

You can create folders or namespaces to organize objects in the model.

The most important thing to know about namespaces is that once you have begun authoring reports, any changes you make to the names of published namespaces will impact your Cognos 8 content. This is because changing the name of the namespace changes the IDs of the objects published in the metadata. Because the namespace is used as part of the object ID in Cognos 8, each namespace must have a unique name in the model. Each object in a namespace must also have a unique name. Part of the strategy of star schema groups is placing shortcuts into a separate namespace, which automatically creates a unique ID for each object in the namespace. This allows us to use the same name for shortcuts to conformed dimensions in different star schema groups.

The next time you try to run a query, report, or analysis against the updated model, you get an error. If you need to rename the namespace that you have published, use **Analyze Publish Impact** to determine which reports are impacted.

Folders are much simpler than namespaces. They are purely for organizational purposes and do not impact object IDs or your Cognos 8 content. You can create folders to organize objects by subject or functional area. This makes it easier for you to locate metadata, particularly in large projects.

The main drawback of folders is that they require unique names for all query subjects, dimensions, and shortcuts. Therefore, they are not ideal for containing shared objects such as conformed dimensions.

Tip: When viewing metadata in the **Diagram** tab, you can expand or collapse folders and namespaces. From the **Diagram** menu, click **Collapse All** or **Expand All**.

If you set security on a folder and then move objects into the folder, confirm that exclusions are set correctly.

For SAP BW metadata, shortcuts to namespaces are not supported.

Steps to Create a Folder

1. From the **Actions** menu, click **Create**, **Folder**.
2. In the **Folder name** box, type a name for the new folder.
3. Click **Next**.
4. Choose whether to move the objects or to create shortcuts:
   - To move selected objects to the folder, click **Move the selected items to the new folder**. When you move an object that participates in a relationship, the relationships to this object also move.
   - To create shortcuts that reference selected objects, click **Create a shortcut for the selected items**. Do not select all the objects in the namespace to avoid creating a recursive structure in the published package.
5. Select the objects you want to add to the folder.
6. Click **Finish**.

Steps to Create a Namespace

1. From the **Actions** menu, click **Create**, **Namespace**.
2. Right-click the namespace, click **Rename**, and give the namespace a descriptive, unique name.
3. Add objects by importing metadata or moving model objects or shortcuts to the objects into the namespace.

Optimizing SAP BW Performance

You can improve performance of SAP BW models and queries by adjusting settings in Framework Manager and in the Cognos 8 studios.

Dimension Settings

For each dimension, check the settings for the **Balanced** and **Ragged** properties.

By default, Cognos 8 uses settings that will not fail for the hierarchy type.

- For dimensions that represent SAP BW characteristics, the **Balanced** property is set to **true** and the **Ragged** property is set to **false**.
- For dimensions that represent presentation hierarchies, the **Balanced** property is set to **false** and the **Ragged** property is set to **true**.

For information about the types of hierarchies, see "Dimensions" (p. 198) and "SAP BW Hierarchies" (p. 193).

Balanced Hierarchies

If a presentation hierarchy is balanced, then the **Balanced** property of its associated dimension should be set to **true**. By default, it is assigned a value of **false** for all presentation hierarchies. A hierarchy is balanced if all leaf characteristic values occur at the lowest level of the hierarchy. By setting this property to **true** as appropriate, the Cognos 8 server is able to generate more efficient MDX.

You can define whether a dimension represents a balanced hierarchy by modifying the **Balanced** property of a dimension. The value that you set depends on the type of object that the dimension represents, and whether the hierarchy is balanced.

When all paths are of the same depth, set the **Balanced** property to **true**, otherwise set it to **false**.

The default value for a dimension representing a presentation hierarchy is **false**.

For a dimension that represents a characteristic without a presentation hierarchy, this property is read-only and is assigned a value of **true**.

Dimension represents	Balanced property value
characteristic	true
presentation hierarchy that is balanced	true
presentation hierarchy that is unbalanced	false
presentation hierarchy whose structure is unknown	false

If you assign a value of **true** to the **Balanced** property of an unbalanced hierarchy, queries that involve this dimension may return incorrect data.

If you assign a value of **false** to the **Balanced** property of a balanced hierarchy, performance may be slower.

Ragged Hierarchies

You can define whether a dimension represents a ragged hierarchy by modifying the **Ragged** property of a dimension. The value that you set depends on the type of object that the dimension represents, and whether you know if the hierarchy is ragged.

By setting the **Ragged** property to **false** as appropriate, the Cognos 8 server is able to generate more efficient MDX.

If a presentation hierarchy is not ragged, set the **Ragged** property of its associated dimension to **false**. By default, set all presentation hierarchies to **true**.

The default value for a dimension representing a presentation hierarchy is **true**.

A dimension that represents a characteristic without a presentation hierarchy is read-only.

Dimension represents	Ragged property value
characteristic	false
presentation hierarchy that is not ragged	false
presentation hierarchy that is ragged	true
presentation hierarchy whose structure is unknown	true

If you assign a value of **true** to the **Ragged** property of an unragged hierarchy, queries that involve this dimension may return incorrect data.

If you assign a value of **false** to the **Ragged** property of a ragged hierarchy, performance may be slower.

Unbalanced Hierarchies

An unbalanced hierarchy can also be ragged. In a ragged-unbalanced hierarchy, there are gaps in the levels and the levels descend to different depths.

Limitations with Ragged and Unbalanced Hierarchies and Aggregated Values

Ragged and unbalanced hierarchies can create gaps within individual paths of a hierarchy, as well as nodes at intermediate levels with no descendants at lower levels. If a report includes query items from a single dimension that are from consecutive or adjacent levels in a hierarchy, the fact values associated with the lower levels should always aggregate to the values associated with the higher levels of the dimension in the report.

However, if the query items are not from consecutive or adjacent levels, and the underlying SAP BW hierarchy is ragged or unbalanced, it is possible that the values of the higher levels may not reflect the aggregation of the fact values from the lower levels. The aggregated fact values associated with the higher levels reflect the aggregated values in the data source. This is typical behavior for OLAP data sources, but may be counterintuitive to those accustomed to reporting against relational data sources.

Data Source Settings

You can use SAP BW variables to reduce and filter the data underlying an SAP BW Query.

If improperly configured, variable prompts that use either a picklist or hierarchical picklist can perform poorly. Use the following properties to control the performance of these types of variable prompts:

- **Level Restriction**

Use this property to reduce the number of characteristic values that populate a hierarchical picklist. There is a limited number of levels of a hierarchy from which values are obtained.

If the value is zero (0), which is the default, characteristic values from all levels of a hierarchy (if applicable to the type of prompt) are used to populate the picklist. Otherwise, the property specifies a colon-separated range of levels from which values are obtained (the root level is zero).

For a ragged hierarchy, you must specify all levels that you may want to use even if some branches do not have that level.

- **Initial Number of Picklist Values**

  Use this property to limit the number of values that initially populate a picklist. The default is 100. It is then under the control of your users to choose the begin and end ranges of values that populate the picklist.

  If the value is set to zero (0), all values are used to populate the picklist.

- **Use Default Value**

  Use this property to set the variable to a single value. Your users are not prompted for the value of a variable and consequently, the Cognos 8 server is not required to populate a picklist with values. However, this removes any flexibility on the part of your users to affect the value of a variable.

- **Prompt Type**

  Use this property to specify the type of prompt.

  Changing a picklist or hierarchical picklist prompt to a type-in prompt can dramatically improve performance because it does not require the application server to populate a picklist with values. Unfortunately, it also requires your users to be able to accurately enter characteristic values.

For information about other variables, see "SAP BW Variables" (p. 223).

Chapter 6: Publishing Packages

You publish a package to make the metadata available to your users. You create packages based on your user groups. Packages must contain all the information that a specific user or group of users needs to create reports.

We recommend that you do the following:

- verify the model and repair any problems (p. 247)
- analyze the model for potential modeling issues that you should examine before publishing the metadata (p. 185)
- set security to restrict access to metadata and data across Cognos 8 products (p. 252)
- specify the languages published with each package (p. 258)
- create or modify a package (p. 250)
- publish the package (p. 261)

In addition, you have the option to externalize query subjects and dimensions to convert them to formats that you can use in Transformer or other applications (p. 259).

If you have published packages, you verify them and repair any problems (p. 247).

Note: Externalizing queries will not be supported in future releases of C8 Business Intelligence Transformer.

Verify a Model or Package

At any point in the modeling process, you can check the validity of the whole model, or selected objects in the model such as a package. We recommend that you first verify the model and repair any problems, and then verify each package within a model individually. By verifying a package, you can find and remove invalid objects that can cause a query to fail.

Verification Categories

When you verify a model or package, Framework Manager looks for messages in the following categories:

Category	Description
Internal Model Inconsistencies	Verifies that objects are properly defined and that duplicate names are not used. We recommend that you always run this group of tests.

Category	Description
Invalid or Incomplete Object References	Checks for dangling references or references to missing objects.
Determinant Completeness	Verifies that determinants are completely defined with keys and attributes, and that all query items are accounted for in a determinant.
Dimension Completeness	Verifies that dimensions are fully defined with level member captions and business keys.
Query Status	Verifies that the evaluation status for query subjects and dimensions is valid and is not set to "needs reevaluation" or "invalid". Query status does not verify shortcut objects. For example, you create a shortcut to an object. The object becomes invalid for some reason. If you verify the object, a message is shown because it is invalid. When you verify the shortcut, no message is shown.
Backward Compatibility	Informs you that a model contains features from a previous release that have been maintained but will be deprecated in a future release.
Verify Relationship Cardinality	Warns when many-to-many relationships are found.

Each category can generate multiple messages for an object. For each message, the severity, object icon, object ID, description of the message, explanation of how to correct the problem, and possible actions that can be performed on the object are provided. You may be able to correct a problem immediately by modifying the object that caused the problem or by asking Framework Manager to repair the problem.

Steps

1. Choose one of the following:

Goal	Action
Verify a model	From the **Project** menu, click **Verify Model**.
Verify a package	In the **Project Viewer**, right-click a package, and click **Verify Selected Objects**.

Goal	Action
Verify selected objects	In the **Project Viewer**, select one or more objects, right-click, and click **Verify Selected Objects**.

2. On the **Options** tab, select the message severity levels that you want to see in the results.

   By default, all message severity level check boxes are selected. Clear the ones that you do not want.

3. Select the categories that you want to verify.

   By default, all category check boxes are selected. Clear the ones that you do not want.

4. Click **Verify Model**.

   The **Verify Model Results** tab shows the results.

5. To sort the messages, click **Sort** in the severity, object type, or name column heading.

6. To see the object that is related to the message, under the **Actions** heading, click **Find in Project Viewer** .

7. To group the messages, select a grouping criteria from the list. If you group the messages, you can repair problems based on the grouping.

8. Select the check box beside each message for the problem that you want to repair.

   Tips:

   - To select all messages, select the check box at the top of the check box column. To clear all selected messages, clear the check box at the top of the check box column.
   - If you grouped the messages, select the check box at the top of the group check box column to select all messages in the group.
   - To hide the message detail information, click **Collapse** in the group heading.

9. Click **Repair Selected**.

   Framework Manager repairs the problems in the following order:

   - invalid references
   - invalid objects, invalid relationships, invalid aggregation rules, missing locales, unsupported prompt types, inferred roles, and re-evaluate; in no particular order
   - upgraded model errors

   You can also repair problems by clicking the **Edit Definition** icon under the **Actions** heading for the message. In the dialog box, modify the definition of the object as required.

When the repair process is finished, a summary of the repair results appears. The model or package is verified again and the results are shown in the **Verify Model Results** tab.

Tip: Problems that cannot be repaired during the verification of a package may be repaired using verify model.

For stand-alone filters, if the underlying objects might not be valid, a "needs reevaluation" message appears. The Repair option does not work for stand-alone filters. To ensure that the stand-alone filters are valid, open each filter and save it.

Create or Modify a Package

You create a package to make metadata available to your users. A package is a subset of a project. It must contain all the information that a specific user or group of users needs to create reports. You can also apply security to the package (p. 252).

For example, if your data source contains information from different areas of a business, you might decide to create different packages for Human Resources and Finance. Ensure that your package meets a broad but related reporting need. Each report can contain information from a single package only.

After a package is published to the server, it is available to your users.

Reusing Packages

You can reuse packages by creating nested packages. When you create nested packages, you create a master package that is based on other existing packages. Using nested packages saves you time, and they are easier to maintain. Another advantage of using nested packages is that you publish only the master package.

For example, you create three separate packages named Canada, Mexico, and the United States. Each package contains the project objects and security appropriate for that package. You can create one master North America package and include the packages Canada, Mexico, and the United States.

You can also reuse packages to create a consolidated package with connections to multiple data sources.

Selecting, Hiding, Unselecting

When you create a package, you can choose whether objects in a project can be selected based on the requirements of your users.

Option	Description
Select	The object can be used in reports and can be selected by your users.

Option	Description
Hide	The data within the object cannot be used in reports because it cannot be selected by your users. Any existing reports referencing this object will not run. For example, you include a model query subject in a package. Because model query subjects are dependent on data source query subjects (p. 87), you must add the data source query subject to your package. If you do not want your users to see the data source query subject, hide it.
Unselect	The object is not published. It cannot be used for reports and cannot be selected by your users.

Including a Model Query Subject in a Package

If a model query subject references other query subjects in a macro or a prompt, ensure that you include the referenced query subjects in the package. This can occur in the following situations:

- A macro for the model query subject references query items in another query subject.
- Another query subject is referenced in the **Prompt Info** properties.

Steps to Create a Package

1. Click the **Packages** folder, and from the **Actions** menu, click **Create, Package**.
2. In the **Provide Name** page, type the name for the package and, if you want, a description and screen tip. Click **Next**.
3. Specify whether you are including objects from existing packages or from the project and then specify which objects you want to include.

   If you created other packages, we suggest that you add package references by clicking **Using existing packages**.
4. Choose whether to use the default access permissions for the package:
   - To accept the default access permissions, click **Finish**.
   - To set the access permissions, click **Next**.
5. Specify who has access to the package, and click **Next**.

   You can add users, groups, or roles (p. 253).
6. Move the language to be included in the package to the **Selected Languages** box, and click **Next**.
7. Move the sets of data source functions you want available in the package to the **Selected function sets** box.

If the function set for your data source vendor is not available, make sure that it was added to the project. For more information, see "Select Function Sets" (p. 317).

8. Click **Finish** and choose whether to publish the package (p. 261).

Steps to Modify a Package

1. Click the package you want to modify and, from the **Actions** menu, click **Edit Definition**.
2. Click the objects you want to add to or remove from the package.

   Tip: To toggle through the options for an object, click the object icon, or select an option from the list. For more information, see (p. 250).
3. Click **OK**.
4. If you want to add or remove package references to the package you are modifying, click **Edit**.

Package Security

In Framework Manager, security is a way of restricting access to metadata and data across Cognos 8 products.

There are different types of security in Framework Manager:

- data security (p. 253)

  You create a security filter and apply it to a specific query subject. The filter controls the data that is shown to your users when they set up their reports.
- object security (p. 255)

  You secure an object directly by allowing users access to the object, denying users access to the object, or keeping it hidden from all users.
- package security (p. 258)

  You apply security to a package and identify who has access to that package.

Each type of security uses users, groups, and roles to define access.

There are business reasons for restricting access to data. For example, you may have confidential data that only specific users are allowed to see. You may have a variety of data, and your users only need to retrieve data from specific tables or columns. Or, you may have a table that contains many records, and your users only need to retrieve a subset of records from that table.

If you are using SAP BW metadata, there can be underlying SAP BW security that affects your users' access to level members. You cannot override SAP BW security in Framework Manager. For more information, see "Import from an SAP BW Data Source" (p. 191).

Before you add security in Framework Manager, ensure that security was set up correctly in Cognos 8. For more information, see the *Administration and Security Guide*.

Users, Groups, and Roles

Users and groups are created for authentication and authorization purposes. You can use users and groups created in third-party authentication providers, or you can create your own in Cognos 8.

For more information about security, see the *Administration and Security Guide*. For more information about users, groups, and roles, see the *Content Manager User Guide*.

Users

A user entry is created and maintained in a third-party authentication provider to uniquely identify a human or a computer account. You cannot create users in Cognos 8.

Information about users, such as first and last names, passwords, IDs, locales, and e-mail addresses, is stored in the providers.

Users can become members of groups defined in third-party authentication providers and groups defined in Cognos 8. A user can belong to one or more groups. If users are members of more than one group, their access permissions are merged.

Groups and Roles

Examples of groups are Employees, Developers, or Sales Personnel. Members of groups can be users and other groups. Group membership is part of the user's basic identity. When users log on, they cannot select a group they want to use for a session. They always log on with all the permissions associated with the groups to which they belong.

A role is a special group. It represents a collection of users that have similar responsibilities and similar privileges in the organization. Members of roles can be users, groups, and other roles. Role membership is not part of the user's basic identity.

You can use groups created by your organization in third-party authentication providers, or create new groups in the Cognos namespace.

Create Cognos groups when

- you cannot create groups in your authentication provider
- groups are required that span multiple namespaces
- portable groups are required that can be deployed
- you want to address specific needs of Cognos 8 administration
- you want to avoid cluttering your organization security systems with information used only in Cognos 8

Add Data Security

You can restrict the data represented by query subjects in a project by creating a security filter. The security filter controls the data that is shown to your users when they set up their reports.

For example, your Sales team consists of a Sales Director, and four Sales Managers. You create a security filter that includes a group for the Sales Director and a group for Sales Managers, and

apply the filter to the Salary query subject. When the package is available for your users, and a report is generated for the Sales Managers and the Sales Director, only the Sales Director can see the salary information for the Sales Managers. For more information about groups, see "Users, Groups, and Roles" (p. 253).

If a user has multiple roles, the security filters belonging to these roles are joined together with ORs. If a role is based on another role, the security filters are joined together with ANDs.

You can base the security filter on an existing security filter. If you choose this option, the security filter inherits the filter and all the filter properties.

When you create a security filter, you can also use existing project filters, or create new filters using the expression editor. For more information, see "Create a Filter" (p. 156).

Steps

1. Click the query subject you want, and from the **Actions** menu, click **Specify Data Security**.
2. To add new users, groups, or roles, do the following:
   - Click **Add Groups**.
   - In the **Select Users and Groups** window, add users, groups, or roles. For information about how to do this, see the *Content Manager User Guide*.
     - In the **Select Users and Groups** window, click **OK**.
3. If you want to base the group on an existing group, click a group in the **Based On** column.

   Tip: If you do not see the group you want in the list, you must add the group to the security filter.
4. If you want to add a filter to a group, in the **Filter** column, click either **Create/Edit Embedded Filter** or **Insert from Model**.

Using the CSVIdentityName Macro Function

If you want row-level security based on UserClass values stored in your data source, implement a parameter map that maps the values in the data source to the corresponding roles and groups based on the user you are logged on as.

You do this by using a parameter map as an argument with the `CSVIdentityName` macro function. This macro function retrieves account, group, and role information for the current user. It returns a string of comma-separated values from the parameter map in single quotation marks, such as 'clerks', 'technicians', or 'typists'.

The `CSVIdentityName` macro function is used as a key in the specified map. You can use the list that is returned to build partial `In` clauses or to filter data based on the identity name of the current user.

For example, you have user classes whose names do not correspond to the Roles_Groups parameter map:

Key (Role or Group)	Value (User Classes)
Everyone	Group1
Authors	Group2
System Administrators	Group3
Query Users	Group2
NTLM	Group2

You have this query subject:

```
(Security_column, value 1, value 2, value 3)
```

When you add a filter to the query subject, the filter uses a macro to look up a list of values, for example:

```
Security_column in (#CSVIdentityName(%Roles_Groups)#)
```

For users in the Everyone, Authors, and System Administrators roles, testing shows this as:

```
Security_column in ('Group1','Group2','Group3')
```

Using the CSVIdentityNameList Macro Function

If security data in the data source is identical to the roles and groups defined in Cognos 8, you can use the `CSVIdentityNameList` macro function. The macro function optionally accepts a list separator as a parameter and then returns a separator-delimited list that can be used in a filter with the In operator. You do not need a parameter map.

Here is an example:

```
Security_column in (#CSVIdentityNameList()#)
```

For users in the Everyone, Authors, and System Administrators roles, testing shows this as:

```
Security_column in ('Everyone','Authors','System
Administrators')
```

Consider the following:

- users can belong to several groups or roles
- there is no way to distinguish between groups and roles so each group and role must have a unique name.
- this function works only in a filter and always returns `0..n` values

Add or Remove Object Security

Metadata security can be applied directly to objects in a project. When you add object-based security, you apply a specific user, group, or role (p. 253) directly to the object. You choose to make the object visible to selected users or groups. You cannot deny the **Everyone** group access to all objects.

If you do not set object-based security, all objects in your project are visible to everyone who has access to the package. Users, groups, or roles that do not have allow or deny settings for an object are considered to be undefined. The object then inherits the security that was defined for its parent object. When you explicitly allow or deny access to an object, you override the inherited settings. The parent and child objects then have different settings. When you apply security to a parent object, all its child objects in the model will also have security applied to them. After you set security for one object, you must set it for all objects. You can do this by setting security on the root namespace.

You may want an object to be visible to only selected users, groups, or roles. For example, in your project, you may have a Salary query subject. You can make the Salary query subject visible to the Managers group, and keep it hidden from everyone else.

If a user is a member of multiple user groups and an object is visible to one user group and denied to the other, the user will not have access to the object. For example, Jane belongs to two user groups: Sales and Marketing. The Sales group has access to the Products and Sales query subjects, and is denied access to the Sales Forecast query subject. The Marketing group has access to Products, Sales, and Sales Forecast query subjects. Jane does not have access to Sales Forecast.

When you secure an object, a package is automatically created in Framework Manager. The package name consists of an underscore (_) and the name of the secured object. These object-based packages are visible in the **Explorer**. You can use this package to see which objects in the project are included, hidden, or excluded from a specific user group.

Every time you include that object in a package, and publish it for your users, the same security rules apply for that object. When you publish a package that contains secured objects, the visible objects for users are the intersection of the package definition and the object security settings. If object-based security is not used, security applied to a package remains unchanged.

Scope of Object Security

The security you specify for an object is passed to shortcuts that reference the secured object. If you have a shortcut to a secured object, only users with permission to see the secured object are able to see the shortcut in the published package.

If a model query subject, calculation, or filter references a secured object, the object's security is not passed to the model query subject, calculation, or filter.

When you create a package containing the shortcut, the secured object does not need to be included in the package.

For example, only sales managers are allowed to see the Sales Target query subject. You create a shortcut to Sales Target. When you package the model, you include the shortcut but not the Sales Target query subject. Sales managers are the only ones able to see the shortcut in the published package.

If your model is segmented, object security is not inherited from the master model. You must define object security on all model segments.

Tips:

- To see a list of the object-based packages, double-click the **Packages** folder. The list appears in the **Explorer** tab. To see which objects are secured against that specific object-based package, click the **Packages** folder, and from the **Actions** menu, click **Packages**, **Explore Packages**(p. 267).
- To determine if object-based security is set in the model, click the **Packages** folder, and from the **Actions** menu, click **Packages**, **Explore Packages**(p. 267). Click the **Roles Explorer** tab. If object-based security was set, you see a package for the Everyone role.
- To determine which objects are explicitly secured in the model, look at the object's icon in the **Project Viewer**. The top left corner of the icon is marked with an overlay.
- To find all objects that were explicitly secured under a given object, select the object and, from the **Tools** menu, click **Find All Secured Objects**.
- To remove object-based security for a particular user, group, or role, delete the package for that user, group, or role from the **Project Viewer**.
- To completely remove object-based security from the model, delete the package for the Everyone role from the **Project Viewer**.

Steps to Add Object-Based Security

1. Click the object you want to secure, and from the **Actions** menu, click **Specify Object Security**.

   Tip: You can select more than one object at a time.

2. Select the users, groups, or roles you want to change. Or, click **Add** to add new users, groups, or roles.

   For more information, see the *Content Manager User Guide*.

3. Specify security rights for each user, group, or role by doing one of the following:
   - To deny access to a user, group, or role, select the **Deny** check box next to the name for the user, group, or role. Deny takes precedence over Allow.
   - To grant access to a user, group, or role, select the **Allow** check box.

   Tip: To allow everyone to see all objects unless specifically denied access, select the **Allow** check box for the Everyone role.

4. Click **OK**.

   A list of new and updated object-based packages appears.

Steps to Remove Object-Based Security for an Individual Object

1. Click the secured object, and from the **Actions** menu, click **Specify Object Security**.
2. Remove security rights by clearing both the **Allow** and **Deny** check boxes for all users, groups, or roles.
3. Click **OK**.

A list of packages that are affected by these changes appears.

Modify Package Security

You can define or change metadata security after a package has been published. To define metadata security the first time you publish a package, see "Publish a Package" (p. 261). You can also modify the security for a package from Cognos Connection.

You can organize your security by specifying which users, groups, and roles have access to certain parts of the published model.

To add metadata security, do the following:

- ❑ Decide whether the objects can be selected, unselected, or hidden in the package (p. 250).
- ❑ Decide which users will have administrative access to a package.
- ❑ Add users, groups, and roles to the package.

When you apply administrative access to a package, you give access to the user or users who are responsible for

- republishing a package in Framework Manager to the Cognos 8 server
- ensuring that no reports are impacted when a Framework Manager package is republished to the server

Steps

1. Click the package you want to edit, and from the **Actions** menu, click **Package**, **Edit Package Settings**.
2. Click the **Permissions** tab.
3. In Cognos Connection, create, add, or remove groups or roles as required. For information about how to do this, see the *Cognos Connection User Guide*.
4. After you finish modifying the security for the package, click **OK** to return to Framework Manager.

Specify Languages

You can specify which languages are published with each package. You can create several packages based on the same model, each using a different language.

For example, the package for the Mexican sales office includes Spanish and English. The package for the Canadian sales office includes French and English.

You can also specify the languages for all packages at one time.

You must add languages to the project (p. 131) before you can specify the languages that your users require in packages.

Steps for One Package

1. In the **Project Viewer**, click the package you want to modify.
2. In the **Properties** tab, find the **Language** property and click **Click to edit**.
3. Click a language (or Ctrl+click multiple languages) in the **Available Project Languages** box and use the arrow button to move it to the **Selected Languages** box.

Steps for All Packages

1. In the **Project Viewer**, click the **Packages** folder.
2. From the Actions menu, click **Packages** and click **Specify Package Languages**.
3. Select the check box for the language you want for each package.

Externalizing Query Subjects and Dimensions

When publishing a package, you have the option to externalize query subjects and dimensions into formats that you can use in Transformer or other applications. Special considerations must be given when externalizing models based on SAP BW metadata. For more information, see "Guidelines for Working with SAP BW Data in Other Applications" (p. 557).

You first define how each object will be externalized by specifying a method to use. When you publish the package, you specify that the query subjects and dimensions are to be externalized.

If you specified a maximum number of rows to be retrieved in the **Governors** dialog box, this setting is ignored.

You have several options for the externalization method.

The Default Method

Use the Default method to specify the objects in a package that you do not want to be externalized. To improve performance, you may not want to externalize all objects in a package.

The CSV Method

Use the CSV method to generate a comma separated file that contains the results of the query subject. In a CSV file, the first row represents the column names and each of the following rows contains one record from the query result set. One file is generated for each query subject or dimension that is set to be externalized.

With the CSV method, you can use locally processed functions to create a dataset for use in Transformer. You can process Cognos SQL locally or on the data source, and capture the result set in a file that can be used in Transformer.

The generated file is restricted to 2 GB in size and contains data based on the native encoding of the current operating system. For example, for Windows 2000, this is specified by the default system locale in the Windows regional settings. For Windows XP and 2003, this is specified by the **language for non-Unicode programs** option in the Windows regional settings.

This option is intended for use only with Transformer. For any other purpose, we recommend that you use the tab method.

To externalize SAP BW query subjects, use the CSV method. For more information, see "Guidelines for Working with SAP BW Data in Other Applications" (p. 557).

The Tab Method

Use the tab method to generate a tab delimited file that contains the results of the query subject or dimension. The generated file can be used directly as a data source. The generated file contains data based on Unicode using UTF-16 LE (Little Endian) encoding with BOM (Byte Order Mark). One file is generated for each query subject or dimension that is set to be externalized.

This method does not work with Transformer because Transformer does not support Unicode. Use the CSV method to create files for Transformer.

The IQD Method

Use the IQD method to generate a query definition file to be used in Transformer. One file with Native SQL is generated for each query subject or dimension that is set to be externalized. The generated file contains data based on the native encoding of the current operating system. For example, for Windows 2000, this is specified by the default system locale in the Windows regional settings. For Windows XP and 2003, this is specified by the **language for non-Unicode programs** option in the Windows regional settings.

The query subject must not require any local processing. It must be able to be run entirely on the data server. We recommend that you test the query subject by setting the query processing for this data source to database only. An error message then appears if the query subject requires local processing.

If you must use locally processed functions to create a dataset, we recommend that you use the CSV method. With the CSV method, you can process Cognos SQL locally or on the data source and capture the result set in a file that can be used in Transformer.

Stored procedure query subjects can be externalized for use in Transformer. The stored procedures must not contain any parameters.

Note that the IQD method will continue to be supported in this release but will not be enhance. For more information, see "Deprecated Features in Version 8.3" (p. 26)

The Externalize Auto Summary Property

You can specify that the output be aggregated or grouped or both. By default, Framework Manager returns rows at the detail level without applying any aggregation or grouping. This property is used when you want to have relational data aggregated when it is externalized. Specify determinants for the query subject before externalizing it.

Use the **Externalize Auto Summary** property to apply the setting of the **Regular Aggregate** property to query items whose **Usage** property is set to **fact**.

If you want to have a specific order of items in the `Group By` clause, specify determinants first, and then set the **Externalize Auto Summary** property.

You can use the **Externalize Auto Summary** property with all externalize methods.

Supported Data Types

Framework Manager supports strings, integers, and dates. It does not support time dimensions. We recommend that you use a date key on the fact query subject in Framework Manager, and let Transformer generate the time dimension.

Shortcuts

If a shortcut is included in a package and it points to a query subject that has been externalized, the shortcut will also be externalized. The name of the data file is the name of the query subject to which the shortcut points. If more than one shortcut points to the same query subject, then the query subject is externalized each time the shortcut is encountered.

Query Processing

Native SQL is used to generate an IQD, so the native SQL produced when externalizing must run successfully.

Some queries cause more than one query to be issued, or local processing to be performed to retrieve data, or both. To prevent this, ensure that the **Query Processing** property for all data source objects in the model is set to **Database Only**. For more information about query processing, see "Improve Performance by Setting Query Processing Type" (p. 315).

Process to Externalize Dimensions

We recommend that you follow this process to externalize dimensions:

- Create a model query subject or a data source query subject that contains the dimensions you want to externalize (p. 87).
- Add any filters that you require.

  For information about filters for relational metadata, see(p. 156)"Create a Filter" (p. 156). For information about filters for SAP BW metadata, see "Create a Filter" (p. 233).
- In the **Properties** pane, set the **Externalize Method** property to the method you want.
- Publish the package to externalize the dimensions you selected (p. 247).

Publish a Package

In Cognos 8, you can publish a package (p. 250) to any folder in Content Manager so your users can access it.

You can also publish a package to a network location. A package on a network location cannot be used by your users. Publishing to a network location is useful for backing up a package.

Objects that are excluded or hidden, but are necessary in a query path, are included but marked as hidden in a published package.

To avoid problems, troubleshoot the package before publishing it by using the **Verify the Package Before Publishing** check box in the Publish wizard to ensure that it is complete and does not contain errors.

When you publish a package, you can

- Set the number of model versions to retain on the server. For more information about model versions, see "Update a Report to Use the Latest Version of a Package" (p. 265).

  Tip: To see the number of model versions set for a package, select a package and, in the **Property** pane, find the **Max Versions** property.

- Externalize query subjects and dimensions so that you can use them with Transformer (p. 259).

Steps

1. Select the package you want to publish.
2. From the **Actions** menu, click **Package**, **Publish Packages**.
3. Choose where to publish the package:
   - To publish the package to the report server, click **Cognos 8 Content Store**, click **open**, and select an existing folder or create a new folder in the Content Store.
   - To publish the package to a network location, click **Location on the network**.
4. To enable model versioning when publishing to the Cognos 8 Content Store, select the **Enable model versioning** check box and type the number of model versions of the package to retain.

   Tip: To delete all but the most recently published version on the server, select the **Delete all previous model versions** check box.
5. Click **Next**.
6. In the **Add Security** window, define security for the package (optional):

Goal	Actions
Create, add, or remove a user, group, or role.	• On the **User Access** tab, click **Add**. • In the **Select Users and Groups** window, define user security. For information about how to use the **Select Users and Groups** window, see the *Administrator and Security Guide*. Users, groups, or roles defined in the user Access Tab have Read, Write, Execute, and Traverse permissions.

Goal	Actions
Grant administrative access to a user, group, or role.	• On the **Administrator Access** tab, click **Add**. • In the **Select Users and Groups** window, define administrator security. For information about how to use the **Select Users and Groups** window, see the *Administrator and Security Guide*. Users, groups, or roles defined in the Administrator Access Tab have Read, Write, Set Policy, and Traverse permissions.

Note: The **Add Security** window in the Publish Wizard is only available the first time you publish a package. To change security for a published package, see "Modify Package Security" (p. 258)

7. Click **Next**.
8. If you want to externalize query subjects, select the **Generate the files for externalized query subjects** check box.
9. By default, the package is verified before it is published. If you do not want to verify your model prior to publishing, clear the **Verify the package before publishing** check box.
10. Click **Publish**.

    If you chose to externalize query subjects, Framework Manager lists which files were created.
11. Click **Finish**.

Publish a Package Based on an OLAP Data Source

You can use Framework Manager to connect to an OLAP data source and create a package based on a cube. You can then publish the package directly to Cognos Connection, making it available for use in the Cognos 8 studios.

By default, each package contains a connection to only one cube. If you want to create a package containing multiple cubes, run the metadata wizard and create a package for each cube. Then create a package that includes individual packages as required.

Before creating a package containing multiple cubes, consider the potential performance impacts. In Cognos Connection, each time a package is used, a connection is made to each of the data sources defined in the package. Creating large packages with multiple cubes can have a negative impact on performance. To offset the potential performance impact of creating one large package containing many cubes, create one package per cube and then create smaller combinations of packages as required.

By default, packages based on an OLAP data source do not contain vendor function lists. If you want to include the vendor function list, specify the function sets to include and then republish the package.

Note: To publish a package that contains a single cube, we recommend that you use Cognos Connection. For information on publishing a package from Cognos Connection, see the *Administration and Security Guide*.

Steps to Create and Publish a Package

1. In the **Welcome** page, click **Create a new project**.

   Tip: If you are already in Framework Manager, click **New Project** from the **File** menu.

2. In the **New Project** page, specify a name and location for the project and click **OK**.

   You may be prompted to provide authentication information.

3. In the **Select Language** page, click the design language for the project.

   The language you select cannot be changed after you click **OK**, but you can add others. For more information, see (p. 135)"Add a Language to a Project" (p. 135).

4. In the **Metadata Wizard** dialog box, click **Data source** and click **Next**.

5. Select your data source from the list of available data source connections and click **Next**.

   If the data source connection is not available in the list, you can click **New** to create the data source connection. For more information, see "Create a Data Source Connection" (p. 61).

6. Specify a name for the package and click **Next**.

   Optionally, you can specify a description and screen tip for the package.

7. Specify who has access to the package.

   You can add users, groups, or roles (p. 253).

8. Click **Finish** to import the metadata and create the package.

9. When prompted, click **Yes** to publish the package or click **No** to return to the **Project Viewer**.

Steps to Add Another Package

1. In the **Project Viewer**, right-click the model that you want to use and click **Run Metadata Wizard**.

2. In the **Metadata Wizard** dialog box, click **Data Sources** and click **Next**.

3. Select your data source from the list of available data source connections and click **Next**.

   If the data source connection is not available in the list, you can click **New** to create the data source connection. For more information, see "Create a Data Source Connection" (p. 61).

4. Specify a name for the package and click **Next**.

   Optionally, you can specify a description and screen tip for the package.

5. Specify who has access to the package.

   You can add users, groups, or roles (p. 253).

6. Click **Finish** to import the metadata and create the package.
7. When prompted, click **Yes** to publish the package or click **No** to return to the **Project Viewer**.

The namespace appears in the **Project Viewer**. You cannot see objects in the native metadata model from within Framework Manager. The native metadata objects are visible from within the Cognos 8 studios when the native metadata package is used.

Publishing a Package by Running a Script

Cognos 8 reports are dependent on the objects in the package on which the report is based. If your report uses objects that no longer exist in the package, the reports will not run. You can have this problem if you make changes to the physical data source, and then use a script to generate the model and republish the package. After the script is run and the package is published, analyzing the impact of publishing the package does not identify the broken reports.

To avoid this problem, do the following:

- ❑ Run the script, excluding any steps that publish the package.
- ❑ Verify the model or analyze the impact of publishing the package.
- ❑ Publish the package.

Update a Report to Use the Latest Version of a Package

When you publish a package for the first time, you create a corresponding package on the Cognos 8 server. The package contains a model but no reports.

When you publish a package, you can select the number of versions of the model to keep on the server. The next time you publish the package, the version of the model is updated in the existing package on the server.

New or modified reports use the latest version of the model in the package. When a report is saved, the version of the model used is saved in the report specification. If the package is republished, the report author is notified that the report uses the newest version of the model in the package. The report author must save the report to complete the update. If you open a saved report after the package it is based on is republished, one of two things happens:

- If the original version of the package still exists, the report runs against the original version. If you want the report to run against the latest version of the package, you must update the report to use the latest version of the package. See the steps below.
- If the original version of the package no longer exists, the report is updated to run against the most recent version.

Step

- Do one of the following:
  - Move one report to the latest version of the model by editing and saving the report.

- Before republishing the model, move all reports to the latest version of the model by selecting the **Delete all previous model versions** check box in the **Publish** wizard.
- Before republishing the model, disable model versioning by setting the model version limit to 1.

Chapter 7: Managing the Project

During the lifetime of a project, data may change and new requirements may appear that require you to update models and data sources.

After publishing your project, you can do the following to manage the content:

- Understand what metadata is in your model (p. 267).
- Implement multiuser modeling (p. 269).
- Administer the metadata (p. 289).
- Synchronize projects (p. 306).
- Control and optimize query behavior (p. 309).

Understanding the Metadata in Your Model

Before making changes to a published model, you can better understand the metadata by doing the following:

- Exploring your projects to see packages and roles in a project (p. 267).
- Viewing the objects in your packages to see where specific objects exist (p. 268).
- Creating documentation about your model to satisfy the specific requirements of your company (p. 268).

Explore a Package

When you have a large number of projects and object-based security in a project, it can be difficult to keep everything organized. You can explore packages to see the packages and roles in a project.

In the **Package Explorer**, you see a list of all the packages (normal and object-based) in a project, as well as the objects that were selected, unselected, or hidden for each package.

In the **Roles Explorer**, you see a list of all the users, groups, and roles in a project, and in which package the object-based security is applied. You can also see whether the objects in the project are hidden or visible to that specific user, group, or role.

Steps

1. Select the **Packages** folder.
2. From the **Actions** menu, click **Package** and click **Explore Packages**.
3. Choose what you want to do.

Goal	Action
View the contents of a package	Click the **Package Contents** tab.
Edit the package	Click the **Package Contents** tab, select the package and click **Edit**. For more information, see "Create or Modify a Package" (p. 250).
View the security for each package	Click the **Object Security** tab and select a package.

4. Click **Close**.

View the Distribution of an Object in Packages

When you view the package inclusion of an object, you see, by package, where that object exists and whether it is selected, unselected, or hidden in that package.

If the object is secured, you will also see the object-based package in which the object exists.

Steps

1. Click the object you want to see, and from the **Actions** menu, select **Edit Package Inclusion**.
2. To edit the package, click **Edit Package**.

   For more information, see "Create or Modify a Package" (p. 250).
3. Click **OK**.

Create Model Documentation

After you model the metadata, you can create an HTML or XML representation of the model that can be customized and printed. This is useful for debugging your model or if your company requires this type of documentation to satisfy process requirements.

When you create model documentation, you can document the entire model or you can select a subset of the model. To document the entire model, you click the top-level namespace. The model documentation shows the selected object and all the properties and children of that object.

You can view, save, or print the report in HTML or XML format. If you are viewing the report in XML format, you may want to customize the report output by providing your own XML schema. You can specify the path to the schema in the ModelDocXSL section of the fm.ini file, located in the *install_location*/configuration directory.

Steps

1. Click the object that you want to document.

Tip: Click the top-level namespace to document the entire model.

2. From the **Tools** menu, click **Model Report**.

   The model report appears.

You can save, print, or change the format of the report.

Multiuser Modeling

You can implement multiuser modeling in Framework Manager by:

- Branching and merging (p. 269)

  If you use branching and merging to manage a multiuser project, each user can modify the same objects in a project at the same time. Each user has a copy of the entire project and can work on any part of it. When the branches are merged back into the root project, all conflicts between the root project and the branches are resolved.

- Segmenting and linking (p. 276)

  If you use segmenting and linking to manage a multiuser project, each user can look at the same parts of a project at the same time. However, you must ensure that each user modifies discrete parts of the project. Use links to allow different users to refer to a project at the same time as another user is working on it.

- Using repository control (p. 280)

  If you use repository control to manage a multiuser project, each user must check out the project, make their modifications, and then check the project back in. Only one user at a time can modify the project.

Branching and Merging Projects

Branching and merging enables multiple users to work on the same model at the same time. To do this, the project owner creates a branch of the root project, which is a copy of the project. A team member can modify the branch as required, independently of the root project. Branches can be merged back into the root project as required. Conflicts between the root project and a branch are resolved during the merge process.

There is no limit to the number of branches you can create. You can create a branch from a branch.

If you use a version control system for your project files, you can also create a branch by checking out the most recent version of the file.

Recommendations for Branching and Merging

Use the following guidelines when branching a project:

- Decide how you want to share the metadata in your organization. For more information, see "Methodologies for Branching" (p. 270).

- Divide the project into logical pieces and branch the project to create the logical pieces. For more information, see "Ways to Branch a Project" (p. 272).
- Communication between team members that are working on various branches is very important. Before making a major change to your branch, talk to the other team members to see how the change will impact their branches. For example, adding objects to a branch will probably not impact other team members but deleting an object that other team members are using in their branches will create a conflict when you merge your branch back into the root project.
- Merge the branches in the reverse order that you created them in. For example, Sean creates a branch from the root project. Susan creates a branch from Sean's branch. When it is time to merge the branches back into the root project, Susan first merges her branch back into Sean's branch and then Sean merges his branch back into the root project.
- Merge branches back into the root project often, after making a few changes to your branch, rather than making many changes and merging only occasionally. This makes it easier to resolve conflicts during the merge process. You can merge all branches back to the root project and then branch the project again. Each team member receives an updated version of the model.
- If you want to branch a project that is under repository control, do the following:
  - Create the root project under repository control.
  - Branch the project.
  - Modify the branch project. Do not use repository commands.
  - Check out the root project.
  - Merge the branch back into the root project.
  - Check in the root project.

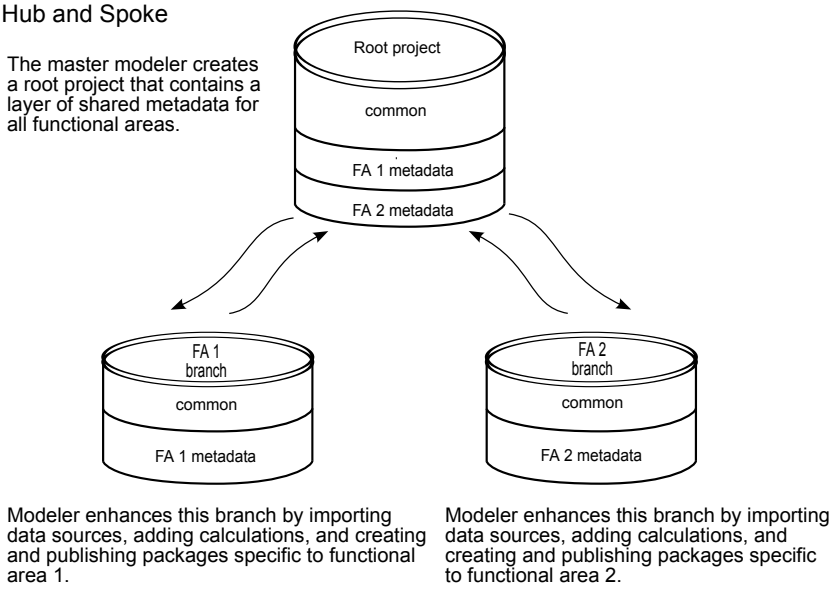
Methodologies for Branching

Before you branch a project, you must decide how you want to share the metadata in your project. This section describes some common methodologies for sharing metadata.

Hub and Spoke

In hub and spoke, a project uses common metadata that must be shared by all functional areas. The root project consists of a fully modeled physical layer containing the objects that all functional areas require. The root project is branched for each functional area. Each functional area can create its own branches if there are multiple people working on it. At any time, a functional area modeler can merge a branch back into the root project to update the root project, and then branch again to receive updates. Objects that are common to all functional areas are kept in the root project.

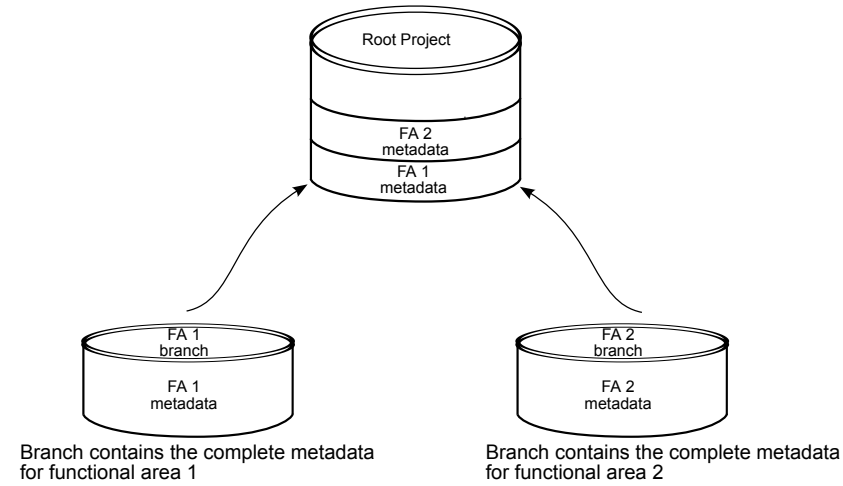
Hub and Spoke


The master modeler creates a root project that contains a layer of shared metadata for all functional areas.
Root project
common
FA 1 metadata
FA 2 metadata
FA 1 branch
common
FA 1 metadata
FA 2 branch
common
FA 2 metadata
Modeler enhances this branch by importing data sources, adding calculations, and creating and publishing packages specific to functional area 1.
Modeler enhances this branch by importing data sources, adding calculations, and creating and publishing packages specific to functional area 2.

Functional Area Specific Metadata

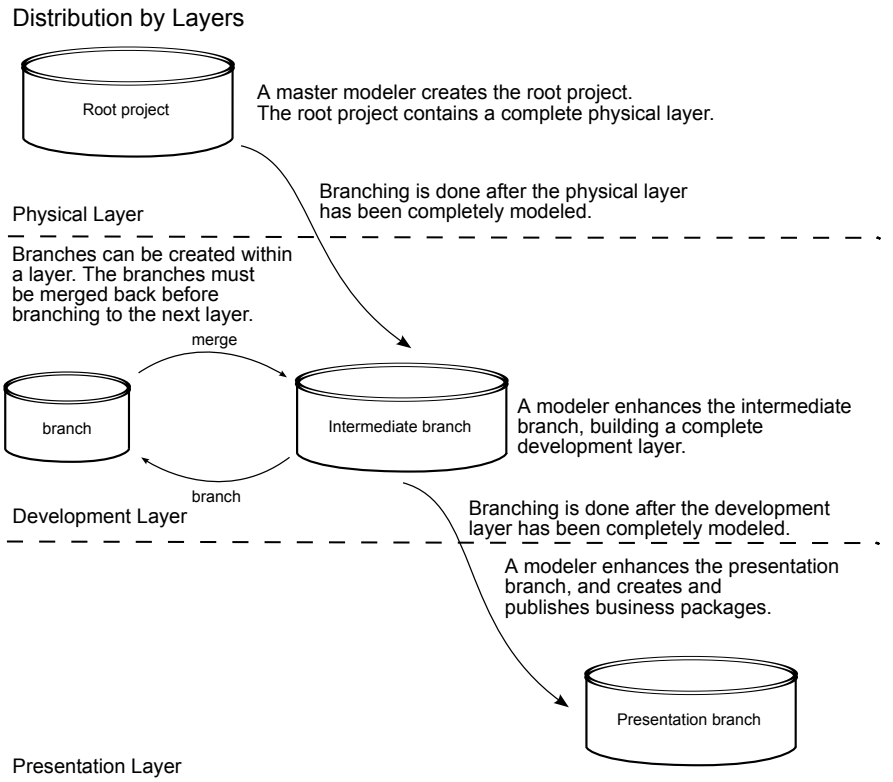
In functional area specific metadata, there is little or no common metadata in the project. Each functional area develops their own objects in the project independently. Each functional area is unaware of the objects in the other functional areas. The master modeler controls merging of the branches to prevent each functional area from seeing objects in the other functional areas.

Functional Area Specific Metadata


Root Project
FA 2 metadata
FA 1 metadata
FA 1 branch
FA 1 metadata
FA 2 branch
FA 2 metadata
Branch contains the complete metadata for functional area 1
Branch contains the complete metadata for functional area 2

Distribution by Layers

In distribution by layers, the metadata is organized in layers. Each layer requires access to the metadata of the layer above it. For example, a model contains three layers. The top layer is the root project, consisting of a fully modeled physical layer. The root project is branched to create the second layer, the intermediate branch. The intermediate branch contains a fully modeled development layer. The intermediate branch is branched to create the third layer, the presentation branch. The presentation branch contains a fully modeled business layer where reporting packages are defined and published.



Ways to Branch a Project

Here are some ways to divide a project:

- By task and skill set

  Create separate branches for one person to work on star schemas and packages, one person to work on calculations and filters, one person to work on dimensions, and one person to work on queries and object naming.

- By model layers

  Create separate branches for the import layer, the intermediate layer, and the presentation layer.

- By business unit or functional area

  Create separate branches for sales and retailers.

- By type of data source

  Create separate branches for relational, dimensional, and ERP data sources.

Create a Branch

You can create a branch in a project.

You can also check out the most recent version of the project file. For more information, see "Check In and Check Out a Project" (p. 285).

Steps

1. Open the project from which you want to branch.
2. From the **Project** menu, click **Branch to**.
3. In **Project name** box, type a name for the project.
4. In the **Location** box, type the path to the folder where you want to store the branched version.

Merge Projects

You can merge a branch back into its root project.

To merge a branch back into the root project, Framework Manager shows a list of transactions that have been performed on the branch. Beside each transaction is a check box. For each transaction that has its check box selected, Framework Manager attempts to perform the transaction on the project with which you are merging. If a transaction is completed successfully, it is marked with a check mark and the next transaction is attempted. If a transaction fails, it is marked with an "X" and processing stops.

When a transaction fails, you may be able to correct the problem immediately. For example, if an object is missing, Framework Manager identifies the missing object and prompts you to replace it with another object.

When you reach the end of the transaction list, you can accept or cancel the changes. If you accept the changes, the transactions that were successful are removed from the transaction list. If you do not accept the changes, the transaction list is not updated and you can perform the same merge again at a later time.

Some conflicts cannot be resolved during the merge process or there may be more than one transaction related to a failed transaction. In these situations, you can skip the transactions by selecting the **Uncheck dependent transactions** check box. Framework Manager then clears the check box for each transaction related to the failed transaction. When Framework Manager resumes running the transaction list, it does not run the transactions if their check boxes are cleared. When you reach the end of the transaction list, you can accept the changes to that point, troubleshoot the transactions that were skipped, fix the error, and then perform the merge again. When you perform the merge again, the transaction list contains only the transactions that were not performed.

Note: Before the transaction list is run, Framework Manager makes a backup of the merging project. The backup is stored in the same directory as the merging project.

Steps

1. Open the project into which you want to merge a branch.
2. From the **Project** menu, click **Merge from**.
3. Locate and click the .cpf file for the branch to be merged.

4. Click **Open**.

   The **Perform the Merge** window opens, showing a list of transactions. The transactions that you selected are run.

5. Choose how to run the transactions:

   - To run the entire transaction list continuously from start to finish, click **Run**.
   - To run one transaction and then pause, click **Step**.

   When a transaction is completed, a check mark or an "X" appears beside it. A check mark indicates that the transaction was applied successfully to the model you are merging into. An "X" means that the transaction failed. Detailed information for each transaction is listed under **Transaction Details**. If a transaction fails, the merge process pauses.

6. Choose one of the following:

Goal	Action
Fix a failed transaction	See "Fixing Failed Transactions" (p. 275).
Skip the current transaction and run the one after it	Click **Skip**.
Run the current transaction and pause	Click **Step**.
Run the transaction list from the current transaction to the end	Click **Continue**.
Accept transactions run to this point and return to the project page	Click **Accept**.
Cancel all transactions run to this point and return to the project page	Click **Revert**.

7. Perform the previous step until you reach the end of the transaction list or you have accepted or reverted the changes.

   If you accepted the changes, they appear in the **Project Viewer**. If you reverted the changes, none of the changes appear.

8. If you accepted the changes, save the merged project in the project page. If you decide not to save the changes, close the project without saving it.

   Note: If you accept the changes in the **Perform the Merge** window but do not save the project you can never perform the same merge again. After you accept transactions they are removed from the transaction list.

Fixing Failed Transactions

During the merge process, transactions may fail. You can fix most transactions by substituting one object for another. You can fix all transactions by modifying the project.

Case 1:

An object used in the branch project is not in the root project. In the **Merge** dialog box, you see the **Replace** box indicating the name of the object that is missing from the root project.

To resolve this problem, you need to select an alternative object from the root project. Do the following:

1. From the **Project Viewer** or **Diagram**, select the name of the object to use in place of the missing object.

   In the **Merge** dialog box, the object appears in the **Replacement Field**.

2. Click **Replace**.

   The transaction runs again, substituting your replacement object for the missing object.

For more complex problems, you may be prompted to select the object that cannot be found in the root project from the **Object Naming Conflict Resolution** dialog box.

If a warning appears, you are unable to resolve the problem using this dialog box. Instead, do one of the following:

- modify the root project (p. 275)
- skip the transaction (p. 275)

Case 2:

For all failed transactions, you can resolve the problem by modifying the root project.

To modify a root project, do the following:

1. Modify the model as required.
2. From the **Perform the Merge** dialog box, click **Continue** to run the entire transaction list starting from the failed transaction.

Case 3:

You are unable to fix the transaction.

To resolve this problem, do the following:

1. Click **Skip** to skip the next transaction.
2. Clear the check box for the failed transaction.
3. Ensure the **Uncheck dependent transactions** check box is selected.
4. From the **Perform the Merge** dialog box, click **Continue**.

Segmenting and Linking Projects

You can use Framework Manager to create and link segments, projects, and folders. A segment is a project within a main project. A segment is owned by its main project. A link is a shortcut to an existing project. The project that is linked can be shared by multiple projects.

A project segment is a complete project and changes to that project impact all projects to which it is linked. If you want to open a segment as a separate project, it must be structured as a complete project. There must be a physical layer in each segment that contains a subset of the data source query subjects on which they are based. These data source query subjects provide access to the data and metadata and must be included in the appropriate segments.

Do not change the physical layer in a segment. Any change will be reflected in the linked parent model and will impact all model segments that share data source query subjects. Changes may not be apparent outside the model in which they are made until the model is closed and reopened.

Before a project is segmented, ensure that the folder and namespace are named correctly. You cannot rename the folder or namespace after it has been segmented.

Changes made in the root model, such as upgrading and setting object security, are not inherited by the segmented model. You have to apply the changes to each segment of the model.

The main project has access to the entire model, including the segments. You can make changes to the segments when working in the main project, however, if the segment is being accessed by more than one user, the potential exists for updates to be lost.

Note: When changing the project structure, do not open the segments as individual projects. We recommend that you check out the main project and make changes from within it.

You can link the segments (p. 279) to other projects that contain related information to maintain consistency and reuse information. If you plan to link model segments, follow the recommendations for segmenting projects. We recommend that you link to relatively complete segments and regularly verify your model (p. 247).

Recommendations for Segmenting Projects

Understanding project segmentation is critical to ensure stability and minimize complexity in a multiuser modeling environment. If you intend to segment your project, we recommend that you do the following:

- Model the physical layer as completely as possible by:
  - ensuring that the namespace in the main project and any links in the project to folders have the same identifier (p. 78)

    For example, you have a main project and a link in the project to a folder. The folder you link to must exist in a namespace that has the same name as the main project. If the identifier in the main project and that of the linked folder are not the same, any relationships, shortcuts, or expressions that reference objects in the link, from the main project, may not work.

  - ensuring all objects in a project have unique identifiers

For example, you have a main project that contains a query subject named NewQS, and a segment in the project. You open the segment, add a new query subject named NewQS, and save the segment. When you open the main project, an error occurs because a query subject named NewQS already exists in the main project.

- updating references in both the main project and segments in the project

  For example, you have a main project and a segment in the project. In the main project, you have a relationship named qs1_qs2 that exists between query subject1 and query subject2. The query subject named query subject 2 is in the segment. You open the segment, rename query subject2 to query subject3, and save the segment. When you open the main project, an error occurs because the relationship qs1_qs2 is broken. In Framework Manager, any object that relies on a reference, such as shortcuts, model query subjects, and expressions are also affected.

- ensuring that the main project and any segments in the project have the same languages

  For example, you have a main project and a segment in the project. In the segment, you defined the languages English and French. You open the main project, add the language Chinese, and save the segment. When you open the segment, an error occurs because the language Chinese is not defined in the segment.

❑ Organize the physical layer using namespaces.

You should create a namespace for query subjects, calculations, and filters that you expect to be necessary for more than one segment.

You should create a namespace for each collection of query subjects that is unique to a planned model segment.

❑ Accept the default project name when creating the segmented project.

The segmented project must be created in a sub-folder within the master project folder. The default project name is the same as the folder or namespace that contains it.

❑ Segment the model (p. 278) for each namespace you created.

❑ Use a source control repository (p. 280) when possible to restrict access and track changes to your projects and segments.

Limitations of Segmenting and Linking Projects

The following limitations apply to segmenting and linking projects:

- You cannot test objects in a segment or linked project if they refer to objects that exist in an unavailable segment.
- You cannot create new objects in a segment or linked project if they refer to objects that exist in an unavailable segment.
- When you link to a project, all referenced objects (namespace objects, data sources, and parameter maps) are linked. Packages are not linked.

- Changes that you make to a child segment are not reflected in the main project, even after doing a refresh (F5). This happens because another child segment linked to the parent or the main project itself is open. Close all child segment projects and then reopen the main project.
- The point at which you create a segment in a project determines the point at which you can see the segment. If you create a nested segment from the main project, you can see the nested segment from the main project. If you open the segment containing the nested segment as a standalone project, you cannot see the nested segment. If you create a nested segment from a segment opened as a standalone project, you can see the nested segment from the standalone project. If you open the main project, you cannot see the nested segment created from the standalone segmented project.

Create a Segment

With segments, you can organize a project according to business rules and organizational requirements, and share and reuse project information.

You create segments at the folder level or the namespace level. You can create a new project in a new folder, complete with its own associated project files.

When a new segment is created, existing parameter maps from the main project are copied to the new segment. After the segment is created, parameter maps are unique to each segment and cannot be shared between segments. For example, if you are working in the main project, you can use a parameter map in a query subject belonging to a segment. However, if you open the segment, the parameter map is not available.

You can access a segment only from the project in which it was created. For example, you open the main project and create a segment (Segment A). Then you create another segment (Segment B) inside Segment A. From the main project, you can see Segment A and Segment B. However, if you open Segment A by itself, you do not see Segment B.

Before you create segments, consider dividing your project into business units. For example, you have a project named Sales. You can create two folders, one named Products and the other named Orders. You can divide the Sales project at the Products folder and at the Orders folder.

Steps

1. Click the folder or namespace you want to divide, and from the **Project** menu, click **Create Segment**.

   We recommend that you accept the default settings for the project name.

2. To rename the segment, in the **Project Name** box, type a different name.

   This does not change the folder name. If you want to rename the folder, you should rename it in **Project Viewer** before creating the segment.

   For ease of use, we recommend keeping the same name for both the folder and the segment.

3. To add the segment to a source control repository, do the following:
   - Click **Repository**.

- In the **Repository Settings** pane, select the **Add to Repository** check box. In the **Connection** list, choose either to use an existing connection, or configure a new repository connection (p. 280).

4. Click **OK**.

   The **Project Viewer** is refreshed and the icons representing the segmented folder or the segmented namespace are shown.

Create a Link

You create links to help organize work across large projects, to maintain consistency, and to reuse information.

For example, the project named Inventory contains the folder named Products. You can create a link from the Sales Products to Inventory Products. If any changes or additions are made to the Inventory Products folder, you will see them in the Sales Products folder.

If you plan to link model segments, ensure that you follow the recommendations for model segmentation (p. 276).

A linked project is shared by other projects. It should not be created in a sub-directory within the master project directory.

You must create the project, folder, or namespace before you can link to it.

The projects you link must have and the same design language the same languages defined.

Steps

1. In the **Project Viewer**, click the project, segment, namespace, or folder that you want to link to.

   Tip: You can create links only to folders, namespaces, projects, or segments.

2. From the **Project** menu, click **Link Segment**.

3. Locate and click the .cpf file of the project that contains the object that you want to link to.

4. Click **Open**.

   - If the project you selected requires upgrading, you are prompted. For more information, see "Upgrading Models" (p. 371).

   - If the project or repository connection uses a mapped drive letter, you are prompted to keep the mapped drive letter or to change it to a UNC path.

     You must choose the UNC path if your project will be shared by others.

5. Choose the project, segment, namespace, or folder to link to:

   - To link to another project, click **Add Project**, locate the .cpf file and click **Open**. Select the project and click **Add**.

   - To link to a segment, click the segment and click **Add**.

6. Click **OK**.

   A new folder appears in the **Project Viewer**.

Leverage a Read-Only Project

You can make a read-only project available for other developers to leverage while protecting the project from unwanted changes.

Steps

1. Create a share that will host the project that is to be protected.
2. Give read-only access to that share for any developer leveraging the project.

   Note: Changing the file properties of the project to read-only will not do the same thing.

Repository Control

Use repository control to help manage your projects in Framework Manager. You control versions of a project to ensure that each version of a file is recoverable. Repository control also ensures that users in a large organization have access to the most recent changes or versions of a project or segment.

You can use Framework Manager to:

- create a connection to a Visual SourceSafe (VSS) or Concurrent Versions System (CVS) repository (p. 280)
- add a new project to a repository (p. 283)
- add an existing project to a repository (p. 283)
- create a local project from a repository (p. 284)

After your repository has been set up and projects have been added, you can use Framework Manager to:

- check in and check out a project (p. 285)
- get the latest version of a project (p. 286)
- view project history (p. 286)

We do not recommend synchronizing projects that are connected to repositories. For more information, see "Synchronize Projects" (p. 306).

Note: The limitations set by the third party repositories are incorporated into Framework Manager. For more information, see the vendor documentation for the third party data source.

Creating Repository Connections

You can use one of these repositories to manage file versions in Framework Manager:

- Visual SourceSafe (VSS)

- Concurrent Versions System (CVS)

In Framework Manager, you can add a project, segment, link, or namespace to one of the supported repositories.

In a multiuser environment, each user needs a Visual SourceSafe (VSS) client installed on their machine.

You can view a list of repositories currently supported by Cognos products on the Cognos support site (http://support.cognos.com). For information about installing and configuring these repositories, see the documentation provided by the vendors.

You must set up the path to the source control system in Cognos Configuration before you create a repository connection in Framework Manager. For more information, see the Cognos 8 Installation and Configuration *User Guide*.

Create a Repository Using Visual SourceSafe

If you use Visual SourceSafe as your repository, connect to it using your Windows user name and password. When you open a project stored in Visual SourceSafe, you can automatically log on using your Windows user name and password.

Steps

1. Start **Visual SourceSafe Administrator**.
2. From the **Tools** menu, click **Create Database**.
3. Specify where you want to create the database.
4. Select the **New 6.0 Database Format** check box, and click **OK** to create the database.
5. From the **Users** menu, click **Open SourceSafe Database**.
6. Locate the srcsafe.ini file, and click **Open**.
7. You can choose to change the database name. Click **OK**.
8. Click the database you just created, and click **Open**.

   A list of the users assigned to the database appears.
9. Ensure that a user with the same credentials as your Windows account exists.

   Tip: To add a user, from the User menu, click Add User and enter the credentials of your Windows account.
10. From the **Tools** menu, click **Options**, and on the **General** tab, select the **Use network name for automatic user log in** check box.
11. Close **Visual SourceSafe Administrator**.
12. Using a text editor, open the srcsafe.ini file, and type the following command at the end of the file:

    Update_No_Change=Update

This command ensures that the project is updated each time there is a check in.

13. Save the srcsafe.ini file and close the editor.
14. Start Framework Manager and, from the **Welcome** page, click **Repository, Connection Manager**.

    If you have Framework Manager open, from the **Repository** menu, click **Connection Manager**.
15. Click **New**.
16. In the **Connection Name** box, type a name for the connection.
17. In the **Type** list box, click **SourceSafe://**.
18. In the **Settings** window, click in the **Value** pane, and browse to the location of the srcsafe.ini file.

    Note: Do not include the filename of the srcsafe.ini filename in the location.
19. Click **Test**.

    If you specified your path using a drive letter, you are prompted to replace it with a network (UNC) path. This is recommended if the repository is to be shared with other computers.
20. Click **OK**.

Create a Repository Using CVS

You can use Concurrent Versions System (CVS) as a Framework Manager repository.

The CVS executable program is configured using the CogConfig utility. A user may maintain their own copy of the CVS executable, or share a common copy accessible by all.

Note: If you are protecting a project for other developers to leverage but not change, ensure that project resides in a separate repository with permissions for that share set appropriately.

Steps

1. Create a directory for the CVS repository.

   An example is C:\cvs_repository.
2. Create a subdirectory named CVSROOT.

   An example is C:\cvs_repository\CVSROOT.
3. To support a multiuser environment, do the following:
   - Right-click the folder containing the CVSROOT directory that is to be shared, and click **Sharing and Security**.
   - In the **Security** tab, you must have a group that includes all users. You can add each user individually.
   - In the **Sharing** tab, set the permissions to Full Control. All users must be assigned Full Control.
4. Start Framework Manager and, from the **Welcome** page, click **Repository, Connection Manager**.

If you have Framework Manager open, from the **Repository** menu, click **Connection Manager**.

5. Click **New**.
6. In the **Connection Name** box, type a name for the connection.
7. In the **Type** list box, click **CVS://**.
8. In the **Settings** window, click in the Value pane, and browse to the location of the repository.

   Do not specify the CVSROOT in the path. The repository location must be different from the location where the new project is stored.

   An example is C:\cvs_repository.
9. Click **Test**.

   If you specified your path using a drive letter, you are prompted to replace it with a network (UNC) path. This is recommended if the repository is to be shared with other computers. Press **Yes** to use the UNC path, or press **No** to keep the drive letter.
10. Click **OK**.

Working With Repository Connections

Repositories help to maintain stability and minimize complexity in a multiuser environment, while ensuring that each file is recoverable. After you have created the connection to your repository (p. 280), you can do any of the following:

- Add a new project to a repository (p. 283).
- Add an existing project to a repository (p. 283).
- Create a local project from a repository (p. 284).
- Check projects in and out of the repository (p. 285).
- Get the latest version of a project (p. 286).
- View the history and restore an object to a previous version (p. 286).

Segmented projects deserve special consideration when used with repositories. For more information, see "Using Repositories With Segmented Projects" (p. 287).

Add a New Project to a Repository

You can add a new project to a repository at the time it is created. For information about creating a new project and adding it to a repository, see "Create a Project " (p. 33).

Add an Existing Project to a Repository

You can add an existing project to a repository.

You must be connected to a repository before you can add a project. For information about connecting to a new repository, see "Creating Repository Connections" (p. 280).

Steps

1. In the **Project Viewer**, click the project that you want to add.
2. From the **Repository** menu, click **Add Project to Repository**.
3. In the **Connection** list box, select the repository connection.
4. In the **Location in Repository** list box, select a location to add the project and click **Select**.
5. To keep the project checked out from the repository, select the **Keep checked out** check box.
6. Click **OK**.

Create a Local Project From a Repository

If you want to use a project that is connected to a repository as the starting point for a new project, you can create a local project from a repository project.

A local project should be created only once. After the project is created, you can pick up changes to the project by getting the latest version (p. 286).

If the original project is checked in to the repository or if it is checked out by you, the new local project is created and checked out of the repository.

If the project is checked out of the repository by someone else, the new local project is created but it is not connected to the repository.

Ensure that the project does not exist on your computer before creating the local project.

Steps

1. From the **File** menu, click **Open**.
2. Click **Repository**.
3. Select the repository connection and the project file in the repository.
4. Select a directory in which to save the new project.
5. Do one of the following:
   - Select the check box to create a local copy of the project and connect it to the repository. If the project is checked out by another user, you are prompted to open the project as a copy.
   - Clear the check box to create a copy of the project on your computer that is not connected to a repository.
6. Click **OK**.

The project is created and a copy of the project is placed on your computer. Before making changes to the local copy, you should ensure that the project is checked out of the repository.

Check In and Check Out a Project

You can also check your project in and out of one of the supported repositories.

You must already have done the following:

- Set up the source control system in Cognos Configuration.

  For more information, see the Cognos 8 Installation and Configuration *User Guide*.

- Created a repository connection (p. 280).

- Either created a new project or opened an existing project, and added the project to a repository.

When you check in a project, the main project file (.cpf), the supporting XML files, and the action log files are checked in. For more information about project files, see "Objects You Will Use" (p. 31).

For information about installing and configuring third party repositories, see the vendor documentation.

If you have a project checked in to a repository and you make a change, you are notified that you must check out the project or segment before you can make the change.

Steps to Check In a Project

1. In the **Project Viewer**, click the project or segment you want to check in.
2. From the **Repository** menu, click **Check In**.

   Tip: To keep your project or segment checked out, select the **Keep Checked Out** check box.
3. In the **Description** box, add a description.
4. Click **OK**.

Steps to Check Out a Project

1. In the **Project Viewer**, click the project or segment you want to check out.
2. From the **Repository** menu, click **Check Out**.

   The icon for the project or segment reflects that the object is checked out.

Steps to Undo Check Out

1. Click the project or segment for which you want to undo the check out.
2. From the **Repository** menu, click **Undo Check Out**.

   You see a warning message about overwriting any edits made to the project.
3. Click **OK**.

Get the Latest Version

When you use repository control in Framework Manager, you retrieve the most recent version of the project (.cpf), model.xml, customdata.xml, and log files. For more information about project files, see "The Project Page" (p. 35).

In the **Project Viewer**, an exclamation mark shown next to a project or segment indicates that the segment is out of date. A segment may be out of date because you synchronized to a previous version or another user checked the project or segment into the repository.

Steps

1. To see which segments are out of date, click **Refresh** from the **View** menu.
2. Click the project or segment you want.
3. From the **Repository** menu, click **Get Latest Version**.
4. Click **OK**.

View History

Every project or segment that is connected to a repository has a detailed record of its history. You can see version information, comments, and details about the history of the project.

The revision history of a project or segment is logged in the repository.xml file. This file is located in the project folder.

Note: We recommend that you do not make changes to a segment that is opened as a separate project. When a segment is opened as a separate project, changes will not appear in the revision history for the main project. If you must make changes to a segment that is opened as a separate project, ensure that the main project and all additional segments are checked in.

You can also return the project or segment to a previous version. Any changes or edits made to the project since the last check in are lost.

Segmented and Linked Models

Linked projects (p. 279) are not impacted by actions performed on the main project or its segments. You must open the linked project as a separate project and perform repository actions directly on it.

If you are using the revision history with segmented or linked models, do the following:

- Always select the main project in the **Project Viewer** before clicking **View History**.
- Ensure that all segments are checked out of the repository before reverting to a previous version. After reverting to the previous version, check all segments back into the repository, starting with the lowest level and working back toward the main project.

  Note: You can revert to a previous version when the segments are checked in, but you will not be able to check out the segments to make further changes.
- Do not sync to a segment version that had a segment created in it. It is better to sync to the root version that was checked in after the segment was created.

You may receive warnings that some segments could not be found. If you accept these warnings, your project will not contain these segments. They can be restored by syncing to a later version.

- Do not delete a segment and then create a new one with the same name. This may break the revision history.

Steps

1. Click the object that you want to view.
2. From the **Repository** menu, click **View History**.
3. If you want to return the object to a previous version, click the row of the version you want, and click **Sync**.
4. Click **Close**.

When you revert to a previous version, segments that were created after the restored version are not deleted from your computer. They are simply disconnected from the main project.

Using Repositories With Segmented Projects

When you work with repositories containing segmented projects (p. 276), there are some things you should consider.

Segments do not need to be connected to the same repository as the main project. When you create a segment, you can choose to add the segment to an existing repository or you can add it to a different repository.

If you have a project or segment checked into a repository and you make a change, you are notified that you must check out the project or segment before you can make the change. If you make changes in a segment, you are notified in the main project that changes were made to the project segment.

Creating a Segmented Project to be Used With a Repository

Segmented projects that are intended to be used with a repository should be created in the following way. Ensure that you follow the recommendations for model segmentation (p. 276).

- ❑ Create the new project and add it to the repository (p. 283).
- ❑ Check out the main project from the repository (p. 285).

  This ensures that projects can be modified by only one user at a time.
- ❑ If there is more than one segment level, check out the main project where the segment is being created (p. 285).
- ❑ Create the desired folder structure.
- ❑ Create the segment (p. 278) and add it to the repository (p. 283).
- ❑ Check in the new segment, and continue checking in each successive main project until the main project is checked in (p. 285).

The segment structure is not saved until main project is checked into the repository.

- ❑ All users should get the latest revision (p. 286) after the main project has been checked in.

Adding a Previously Segmented Project to a Repository

Follow this process to ensure stability when adding a segmented project to a repository. Ensure that you follow the recommendations for model segmentation (p. 276).

- ❑ Add the main project to the repository (p. 283) and keep it checked out.
- ❑ Add the segments to the repository (p. 283), starting with the main project and working out to the lowest level.
- ❑ Check the lowest level segment into the repository (p. 285), and continue checking in each successive main project until the main project is checked in.

  The segment structure is not saved until main project is checked into the repository.

Remove a Segmented Project From a Repository

Follow this process to ensure stability when removing segmented projects from a repository. Ensure that you follow the recommendations for model segmentation (p. 276).

- ❑ Check out the main project from the repository (p. 285).

  This ensures that projects can only be modified by one user at a time.
- ❑ If there is more than one segment level, check out the main project containing the segment to be removed.
- ❑ Delete the segment (p. 292).
- ❑ Check in each successive main project until the main project is checked in (p. 285).

  The segment structure is not saved until the main project is checked into the repository.
- ❑ All users should get the latest revision (p. 286) after the main project has been checked in.

Working With Segments in Multiuser Modeling

Consider the following when working with segmented projects in a multiuser project:

- Only a single user should change the segment structure by adding or removing segments. When changing the structure of the project, you should always check out the main project.
- When working with a segmented project, check out only the segment you are working with. The main project does not need to be checked out but you should open the main project and work with the segments from within it.
- When checking a segment into the repository, include a concise comment that starts with the name of the segment. This will make the revision history easier to understand.
- When a segment has been checked into the repository, the main project will be shown with an exclamation mark indicating that it is out of date.

Tip: To see which segments are out of date, click **Refresh** from the **View** menu.

- Checking out a project that is out of date, or using the **Undo Checkout** action, requires that all child segments be updated.

  You may receive a message indicating that changes in the child segments will be lost. You must check in any modified child segments before checking out the project or using the **Undo Checkout** action.

Branching a Project that is Under Repository Control

For information about how to branch a project that is under repository control, see "Branching and Merging Projects" (p. 269).

Using External Repository Control

You can use Framework Manager with an external source control system without using the repository features provided by Framework Manager. This procedure explains how to put the Framework Manager project files into an external repository. You can work on the project in Framework Manager and the external repository can manage the version control of the project files.

To use an external source control system, do the following:

- ❑ Ensure that the project is closed.
- ❑ Delete the repository.xml file, if it exists.
- ❑ Add all the files that exist in the project directory to the external repository.

  Tip: The project directory is the directory that contains the <project name>.cpf file.
- ❑ Check the project files out of the external repository.
- ❑ Work on the project in Framework Manager.
- ❑ Save your changes.
- ❑ Check the project files into the external repository.

Segment a Project

You can create a segment for a project that is stored in an external repository. The segments are project directories that are stored under the main project directory. Maintain the same hierarchy in the repository as in the project directory.

The segments can be opened individually as stand-alone projects.

A segment can also be opened as part of the main project. In this situation, you must check out the project for each segment that you want to modify.

Administering the Metadata

You can change the metadata in your models to meet your specific modeling goals.

You can do the following:

- Copy, move, rename, or delete your projects to organize them in meaningful ways for your reporting environment (p. 290).
- Analyze the changes made to a model to see how they affect the packages and the reports that use the model (p. 292).
- Remap an object to a new source (p. 295).
- Export your model to exchange metadata between different data warehouse tools and repositories (p. 297).
- Reuse the same model and reports with different sets of data (p. 298).
- Move a model from one relational database to another (p. 299).

Copy, Move, Rename, or Delete a Project

You should organize projects in a meaningful way so that you can easily find them. Within Framework Manager, you can copy (p. 290), move (p. 291), rename (p. 291), and delete (p. 292) projects.

You can manage your projects using repository control (p. 280), segmenting (p. 278), and linking (p. 279). These project management features help maintain version control, organize a project according to business rules and organizational needs, set run time processing options, and give other users access to sections of the project.

A project that is connected to a repository loses all repository information when you copy, move, or rename the project. Saving a project as a new project also removes all repository information.

You can also identify the vendor-specific functions (p. 317) that you want to use for each data source you import into your project.

If your project is segmented, the segments are treated as standalone projects. If you save or copy a project within an existing project, it is treated as a segment.

Copy a Project

When you copy a project, you create a replica of that project in another location. All files in the project folder, including sub-folders, are copied to the new location. When you make changes to the project in one folder, these changes are not reflected in copies of the project in other folders.

Copying a segmented model copies all segments as well as the main project.

There may be times when you cannot copy a project and must use **Save As** instead. Saving the project with a new name creates a new project folder while saving the project with the existing name overwrites the current project. This is useful if you want to save changes made to a read-only project or if you want to save a project with a different name or to a new location without overwriting the original project.

You cannot create a copy of a project in the same folder as the original. If you copy a project under an existing project folder, Framework Manager treats it like a project segment (p. 276).

If a project or segment is open when you copy it, the last saved version is copied.

Steps

1. From the **File** menu, click **Manage Projects**, **Copy**.
2. In the **From** box, click the browse button and select the .cpf file for the project you want to copy.

   Note: The project folder name is shown in the text box.
3. In the **To** box, type the project name.

   By default, the project name and the directory where the project is saved are the same.
4. In the **Location** box, type the new location or click the browse button and select the new project location.
5. Click **OK**.

Move a Project

You may decide to move a project if your folder becomes so full that it is difficult to locate particular projects. When you move a project, you are actually copying it to a new folder and deleting it from the current folder. All files in the project folder, including sub-folders, are moved to the new location.

Moving a segmented model moves all segments as well as the main project.

Before you can move a project, the project must be closed in Framework Manager.

Steps

1. From the **File** menu, click **Manage Projects**, **Move**.
2. In the **From** box, click the browse button and select the .cpf file for the project you want to move.

   Note: The project folder name is shown in the text box.
3. In the **To** box, type the new location or click the browse button and select the new project location.
4. Click **OK**.

Rename a Project

When you rename a project, you provide a new name for the .cpf file. You are not changing the location of the project. Secondary project files and log files keep their original name.

If a project appears in the recent projects list on the Framework Manager **Welcome** page and you proceed to rename it, you cannot open the project by clicking the link. You must open the project using the **Open** command from the **File** menu.

Before you can rename a project, the project must be closed in Framework Manager.

Steps

1. From the **File** menu, click **Manage Projects**, **Rename**.
2. In the **From** box, click the browse button and select the .cpf file for the project you want to rename.

   Note: The project folder name is shown in the text box.
3. In the **To** box, type the new name for the project and click **OK**.

If the original project folder and .cpf file have the same name, both the folder and .cpf file are renamed.

Delete a Project

When you delete a project, the project folder and all its contents, including any user files, are deleted from the file system and sent to the recycle bin.

If your project is segmented and you delete the main project, the segments are deleted as well. Deleting a project segment deletes only the segment and not the model it is based on.

We recommend that you delete segments from within the model. If you delete the segment using **Delete** from the **File** menu, it appears as if the segment still exists within the model. For more information, see "Segmenting and Linking Projects" (p. 276).

Before you delete a project, ensure that the project and all its segments are closed. Framework Manager does not support a file locking mechanism so it is possible under certain circumstances to delete a project with open segments. If you delete a project with open segments, the segments can no longer be saved.

Steps

1. From the **File** menu, click **Manage Projects**, **Delete**.
2. In the **Project Folder** box, click the browse button and select the .cpf file for the project you want to delete.

   Note: The project folder name is shown in the text box.
3. Click **OK**.

The project folder and all its contents are deleted.

Analyze the Impact of Changes to a Package

Before publishing packages and running reports, you can see how the changes you make to a model will affect the package and the reports that use it. You can find the changes that were made to the package, and see details about each change and which reports are affected by a specific selected change.

Reports that are created using the package may be impacted by changes that you made to the model. For example, adding new objects to a package does not affect a report. Changing the name of a query item does affect a report. The report definition will not be valid because the query item is not in the package definition.

Note: Because a report uses a published package, if you make changes to the model but do not publish the package that uses it, the report is not impacted by the changes.

If you change the name of an object, it shows up as "modified" in the results of the analysis.

The analysis is done on objects that a model uses directly, not the underlying objects. For example, you have a model query subject that is based on a data source query subject. If you change the model query subject, it will show up as a modified object. If you change the data source query subject, it will not show up as a modified object.

The following types of objects are analyzed: query subjects, query items, measures, regular dimensions, measure dimensions, hierarchies, levels, stand-alone filters, and stand-alone calculations.

Steps

1. In the **Project Viewer**, click a package that has been published.
2. From the **Actions** menu, click **Package**, **Analyze Publish Impact**.
3. Choose what you want to do:

Goal	Action
View report dependencies	See Find Report Dependencies.
View the dependencies for an object	See Show Object Dependencies.
See the details for an object	Click the row that contains the object. The details for the object appear under **Change Details for**.
Find an object in the **Project Viewer**	In the row that contains the object, under **Actions**, click **Find in Project View**.
Sort the results	Click **Sort** at the top of a column.
Display modeler's comments, last changed by, and last date changed	Click the double down arrow.

4. Click **Close**.

Find Report Dependencies

You can find the reports that use an object.

Steps

1. From the **Analyze Publish Impact** dialog box, do one of the following:
   - Select each object for which you want to determine the report dependencies by selecting individual check boxes.
   - Select all objects by selecting the check box at the top of the check box column.
2. Click **Find Report Dependencies**.
3. Specify the scope of the search:

Goal	Action
Search all folders	Click **All Folders**.
Restrict the search to a specific folder	Click **Restrict Search** (**Browse and select a folder**). Type the name of the folder or click **Browse** to search for a folder.

4. Click **Search**.

   A list of report names appears in the **Report Dependency** window under **Impacted Reports**.
5. To sort the results, click **Sort** at the top of a column.
6. Click **Close**.

Show Object Dependencies

You can find objects that depend on other objects, or show the dependencies of a child object.

Constraint

You cannot show dependencies for parameter maps.

Steps

1. In the **Project Viewer**, click an object.
2. From the **Tools** menu, click **Show Object Dependencies**.

   The objects that depend on the selected object appear under **Dependent objects**.
3. To show the object identifier for the dependent objects, select the **Show Object ID** check box.
4. If the object has children and you want to see the dependencies for a child object, click the plus sign (+) beside the object that contains the child object.

5. Click a child object under the parent object.

   The objects that depend on the child object appear under **Dependent objects**.

Note: You can also show object dependencies from the following:

- The **Project Viewer** by right-clicking an object and selecting **Show Object Dependencies**.
- The **Context Explorer** window by right-clicking an object and selecting **Show Object Dependencies**.
- The **Analyze Publish Impact** window by clicking the **Show Dependencies** icon under **Actions** in the row that contains the object.

Remap an Object to a New Source

During the life cycle of a Framework Manager model, you may need to change the data source it uses. For example, you may want to use the model against a different database with the same data, migrate the model from a transactional schema to a star or snowflake schema, or replace a previously existing database or import view with a new view. All of these actions can affect your reports. For example, if you change the names of objects, reports may no longer validate.

You can minimize the effect of model changes and data source changes by remapping higher level model objects so that they continue to run and return correct data. When you remap, you match and substitute object references or names in an original object, to object references or names in another object. You can remap query items and measures. You can remap individual objects manually or you can remap multiple objects at the same time. When remapping multiple objects, Framework Manager matches items in the original object to items in the other object using the matching criteria you specify. Only the objects that meet the matching criteria are remapped. You can use the object name or the object reference as the matching criteria for both the original and other objects.

If a model query subject or model dimension contains a filter or calculation, the model filter or calculation is also remapped when you remap the model query subject or model dimension. You do not see a message or warning about this.

Note: We recommend that you validate all affected reports whenever you make changes to your model. To identify affected reports, see "Show Object Dependencies" (p. 294) and "Find Report Dependencies" (p. 294).

Constraint

You cannot remap data source query subjects or data source dimensions.

Steps

1. In the Project Viewer, right-click an object and select **Remap To New Source**.
2. If you want to change the matching criteria, click **Options** and do the following:
   - Choose the matching criteria for the object you are using to remap, and for the original object that you are remapping.

     You can match objects by name or by object reference.

- The default criteria options are **By Name** for the object you are using to remap, and **By Object References** for the original object that you are remapping.
- If you want to set these choices as the default, select the **Set as default** check box.
- Click **OK**.
- To use the criteria you specified, select the **Use matching criteria options** check box.

Notes:

- If the matching criteria is **By Name** to **By Name**, spaces within the string are removed.
- If there is no object reference, the object name is used.

3. Do one or more of the following:

Goal	Action
Remap an individual object manually	Under **Available Model Objects**, drag an object to the object that you want to remap under **Query Items, Measures, Calculations, and Filters**. The new value for the object appears under **Remap To**.
Remap multiple objects automatically	Under **Available Model Objects**, drag a query subject to any row under **Query Items, Measures, Calculations, and Filters**. All of the objects that meet the matching criteria are remapped and their values appear under **Remap To**.
Change the expression for an object	Click the ellipsis (…) button beside the object. For information on how to create an expression, see "Create a Calculation" (p. 153).
Restore a remap value to the original source value	Right-click the row that contains the object that you want to restore, and select **Restore to Original Value**.
Clear the remap value and the original value for the selected object	Click the row that contains the object, and click **Clear**.
Clear the remap value for all objects	Click **Clear All**.

4. Click **OK** when you are finished remapping.

Export Metadata

You can export your Framework Manager model as a Common Warehouse Metamodel (CWM) file. CWM exchanges metadata between different data warehouse tools and repositories. Each instance of the CWM metamodel is exchanged using XMI (.xml metadata interchange) documents.

When you export a Framework Manager model as a Common Warehouse Metamodel (CWM) file, joins, folders, namespaces, prompts, and calculations are not exported. Only query subjects, query items, and functions are exported.

When you export to CWM, we recommend that you use the default options, which optimize the metadata export. Only change these options if you have specific information that affects your export. For more information about export options, see the Meta Integration® Web site.

Constraint

Do not use Japanese characters in the export path.

Steps

1. Right-click the root namespace of the metadata you want to export, and click **Export Model**.

   You are prompted to save the project.

2. Select the export target.

3. In the **File Name** box, browse to locate the file that contains the appropriate metadata type and click **Save**, and then click **Next**.

   An example is Sales_cwm.xml.

4. In the **Framework Manager Specific Export Options** dialog box, click the options you want.

   Note: We recommend that you use the default options. These default options optimize the third party metadata import. If you change the options, you may see unexpected results. To revert to the default options, click **Use Defaults**.

5. Click **Next**.

6. In the **Third Party Specific Export Options** dialog box, click the options you want.

   In the **Option Description** pane, you see a description of the options available. The options are based on the selected third party data source. For more information, see the third party data source vendor documentation.

   Note: We recommend that you use the default options. These default options optimize the third party metadata export. If you change the options, you may see unexpected results. To revert to the default options, click **Use Defaults**.

7. Click **Next**.

   The input validation results from the export process appear.

8. Click **Next** and click **Finish**.

Project Reuse

You may have to use the same model and reports with different sets of actual data. The data sets may be different databases, accounts, or schemas in a single database. You may encounter multiple data sets

- when you use a different data set than used in production
- in large enterprises, where each division has it own data set
- in OEM applications, which have no direct control over customer data

The tables and columns used by the project must be logically the same across all data sets. You must also ensure that the correct data set is identified in each case.

Data sources in Framework Manager contain information that identifies the location of any data source tables needed for the query subjects. This information is the name of the data source in the content store, as well as the optional catalog and schema names. Ensure that the catalog and schema names use the desired data set.

If different content stores are in use, and a different version of the project is deployed to each content store, you can specify the data source information in the project for each site. If you have only one content store, you can publish each project as a separate package. These solutions require a lot of manual maintenance. To reduce this level of maintenance, you can use one of the following options.

Determining the Data Source Information

The simplest solution is to determine the name of the data source in the content store, the name of the catalog, if applicable, and the name of the schema in the database. You can then use these names in all the data sets.

If some data sets use the same content store, create a separate connection for each data set in a single content store. For more information, see the *Administration and Security Guide*. For information about how Framework Manager handles multiple connections, see "Multiple Data Source Connections" (p. 58).

Because the data source name in the content store can differ from the name of the customer database, this solution offers a lot of flexibility. However, it still requires that the catalog and schema names be identical across all data sets. Even if all the data sets use the same database type, this may be difficult to ensure. If different database types are involved, it may be impossible. For example, SQL Server has a catalog level, but Oracle does not.

Using User-based Default Data Set Identification

Each database user has access to a default schema and catalog, if applicable. If the schema and catalog are not defined, or if they are blank in the Framework Manager project data source, the default is used. As in the previous solution, this option may be combined with multiple connections so that different users can use different databases for the same data source.

However, when you edit a query subject, Framework Manager uses the catalog and schema names in the data sources to match them to items that are dragged to the SQL windows from the data

source tree. For this reason, the catalog and schema names cannot be blank in the project data source while you are modeling.

Therefore, you must use a macro expression in the catalog and schema of each data source in the project. This ensures that the catalog or schema names are blank at run time, but explicitly sets the catalog or schema you want while modeling.

Steps

1. Create a single session parameter whose value identifies whether you are in design mode. When you are in design mode, set the value of this session parameter to a specific value, such as design. Otherwise, leave the value empty.

   Tip: If you use a project or override value, you must set it each time you open the model for editing.

2. For each catalog and schema in each project data source, create a parameter map that contains
   - an empty default value.
   - a key whose name is the design value of the session parameter above, and whose value is the name of the design mode catalog or schema for that data source.

3. Select the data source, and replace the catalog and schema property values with a macro that uses the corresponding parameter map and session parameter.

   For example, use

   ```
   #$DBSchemaName ($DeployOrDesign) #
   ```

Model Portability

You can use a Framework Manager model to access data from different database instances. The database instances can be from the same or different vendors.

There are several things to consider when moving a Framework Manager model from one relational database to another. Unlike changing from one identical database to another on the same platform, it may not be sufficient to change the data source connection information.

Review the generation of determinants and relationships based on indexes and do not assume that the indexes reliably describe functional dependencies or relationships for reporting.

Scalar functions are imported into a model prefixed by a catalog or schema qualification in the SQL statement. As with tables and views, you may have to remove or alter the location qualification when switching vendors. For example, if you create a model against an ORACLE database, and the connection is changed to point to an equivalent SQL Server database, an error results because the model data source type has remained OR instead of changing to the appropriate data source type.

To move a model from one relational database to another, do the following:

- Evaluate the DDL (Data Definition Language) to determine portability for physical names by
  - constraining physical names to a lowest common denominator, such as 31 characters.
  - avoiding using reserved key words in the ANSI standard and vendor documentation.

- avoiding using vendor specific data fields.
- avoiding conversions.
- confirming that precision and scale is supported across all vendors.
- using consistent and compatible collations.
- using a consistent case on names, such as all lowercase.

- Evaluate the DDL to determine portability for database qualification.
- Evaluate the DDL to determine portability for data types in terms of compatibility and the precision and scale of data types.
- Review any native SQL statements in your models and reports for relational-specific syntax that may or may not be supported.
- Review usage of vendor-specific functions.

  There may not be an equivalent vendor function or common function. A common function that is unsupported by the relational database may result in local processing that did not previously occur.
- Review the data source properties type.

  If you change the RDBMS you use, such as from Oracle to SQL server, change the type property for the data source in Framework Manager.
- Update the data source queries.

  When you import tables, Framework Manager imports physical information about the tables and columns that is used internally at run time. For example, collation information is reconciled only by rebuilding the physical tables.
- Test the moved model.

  There will be other differences, such as performance characteristics, how data is ordered based on collations, and so on, that are revealed only by testing.

Moving a Model to Another Environment by Using Log Files

In Framework Manager, you can view (p. 301) and play back actions (p. 301) performed on the project or you can use Script Player (p. 302) to play back transactions in batch mode. An action log is an XML file that contains a set of transactions. Each transaction has a sequence number and one or more actions. Each action is made up of a name and input parameters. Some actions also have output parameters. The action log file is in the project folder.

For example, you make changes to a project in a test environment. When it is time to move the project to production, you can use log files to play back every action, or series of actions, that you performed in the test environment to create an identical project in the production environment.

There are two action log files. The `log.xml` file contains all the transactions that have been run and saved in the project. This file is created the first time you save the project and exists until you

delete the project. The temporary file contains transactions that have been run during the current session, but not saved. The temporary file is deleted when you close the project.

Note: If the script has dependencies on the existing project, you must ensure that the project is aligned with the script transactions to ensure the desired results.

View and Save Transaction History

You can view the transaction history in an action log file and then save it as a script.

Steps

1. From the **Project** menu, click **View Transaction History**.

   Tip: To make the dialog box larger, double-click the caption. Double-click again to restore the dialog box to its original size.

2. Click the transaction numbers that you want.

   Tip: To view the details of a transaction, click the plus sign (+) next to a transaction number.

3. Click **Save as Script**.

4. Type a name for the file.

5. Click **Save**. Do not save the file in the **logs** folder.

6. Click **Close**.

Play Back Transactions From a Log File

You can choose to play back a specific transaction or a combination of transactions in a project or segment action log file.

When you play back transactions from a log file, the script player applies the commands in the log file to the contents of the existing model. Errors appear if objects created by the log file already exist in the model.

After the script in a log file has run successfully, a backup of the original project is created in the parent directory of the project. If you want to undo the transactions performed in the script, you can use the backup to restore the project to its original state.

You must disable or clear any commands that will conflict with the contents of the model. You can then run the script again. Or, you can use the **Synchronize** command, which begins with an empty model.

If you generate your own script outside Framework Manager, time stamps must be in ascending order with no duplicates.

Steps

1. From the **Project** menu, click **Run Script**.

2. Select the script you want, and click **Open**.

3. If you want to view the details of a transaction, click the transaction.

4. Set the starting or stop point that you want.
   - To set the starting point for running the script, select the script and then click **Set the starting point**. You can do this at any time to skip an instruction or run instructions that have already been executed .
   - To set a stop point for the script, select the script and then click **Set the stop point** .

     You can stop the script to make a manual fix and then start it again.

     Tip: To remove the stop point, click **Remove the stop point**

     .
5. Using the toolbar buttons, choose the run action that you want.

Button	Description
	Runs the script After an error is encountered, clicking this button attempts to re-execute the failed instruction.
	Skips to the next transaction and runs the script to the end
	Runs the selected transaction only
	Skips to the next transaction and stops, but does not run any transactions

   The project window is updated as the script is run.
6. Fix any errors encountered by the script either by retargeting objects or modifying the temporary project as required.

   For more information, see "Fixing Errors Caused by Invalid Objects" (p. 305).
7. When the script has completed, click **Accept** to accept the changes or click **Revert** to undo the changes.

   Note: After clicking **Accept** or **Revert**, you cannot use **Undo** and **Redo** for the current session.

Running Action Logs in Batch Mode

The Script Player is a command line utility that runs previously created action logs in batch. The Script Player interfaces with the FM engine and metadata directly, allowing you to bypass the Framework Manager application. You can use action logs that have been created either by the Framework Manager application or manually by using the reference material. It is ideally suited for automated tasks related to model creation and model maintenance. After careful analysis of your log files, you can use the options to run specific transactions.

The Script Player reads the command line parameters and either creates a new model or opens an existing one. After that, an action log consisting of a set of transactions is analyzed and each action

is executed in sequence. Actions usually change the model. Before the Script Player terminates, the modified model is saved and replaces the original model. Using the options, you can suppress saving the model. This is useful if you are testing action logs. The Script Player maintains an internal log of all executed actions, allowing you to back out of any changes if necessary.

We do not recommend using action logs to upgrade models. To upgrade a model from ReportNet to Cognos 8, you should open the model in Framework Manager and verify it, choosing the best method to fix reported errors.

You can use the Script Player on UNIX platforms, where the Framework Manager application is not supported.

Syntax

At the command prompt, ensure you navigate to the installation location of the BmtScriptPlayer.exe.

Use the following syntax to run the Script Player:

```
BmtScriptPlayer [-c|-m] <projectname> [-a <actionlogname>][options]
```

where <projectname> is the name of the project and <actionlogname> is the name of the action log.

For example,

```
BmtScriptPlayer -m goSales.cpf -a import.xml
```

Options

You can specify how the Script Player runs using the following options.

Note: If you are working in a UNIX environment, you may want to create a script to hide credentials that are passed on the command line.

Option	Description
`-a FILEPATH`	Apply the specified action log. FILEPATH is the path, including the file name, to the action log file.
`-b NUM`	Execute transactions with sequence number equal to or higher than the number specified by NUM. The default is the first transaction.
`-c FILEPATH`	Create a new project. FILEPATH is the path, including the file name, to the models project (.cpf) file. Using this option without specifying an action log results in the creation of an empty model. **Note:** If the model specified in the FILEPATH already exists, it is silently replaced.

Option	Description
`-e NUM`	Execute transactions with sequence number equal to or lower than the number specified by NUM. If the option is not specified, execution ends at the transaction with the highest sequence number or transaction number 9999, whichever comes first. For action logs that contain transactions with sequence numbers 10,000 and higher, this option must be used.
`-g`	Upgrade the model (if required). If this option is not specified and the model was created with a previous version, execution terminates. If you specify this option without specifying an action log, only the model upgrade is performed.
`-k DIRECTORY`	Specify the install directory.
`-m FILEPATH`	Open an existing project. FILEPATH is the path, including the file name, to the models project (.cpf) file.
`-n`	Do not save the model. This option can be used to test action log files.
`-p PASSWORD`	Authenticate using the specified password (if required).
`-s NAMESPACE`	Authenticate using the specified namespace (if required).
`-t DIRECTORY`	Specify the template directory.
`-T PASSPORT`	Specify a security passport. A passport is an encrypted string used to allow secure conversations for the plug-ins that need it.
`-u USER`	Authenticate using the specified user name (if required).
`-x`	Terminate the test run when there is a transaction error. By default, the script player only terminates with severe errors such as an invalid model or action log, and continues executing, even if some minor transactions fail.

Option	Description
`-y PASSPORT`	Authenticate using the specified passport (if required). This option overrides other specified credentials (-s, -p, and -u). The Script Player skips authentication and associates the specified passport with the session.

Examples

This table shows some examples of Script Player commands.

Command	Description
`BmtScriptPlayer -c <projectname>`	Create a project.
`BmtScriptPlayer -c <projectname> -a <actionlogname>`	Create a project and apply all the transactions from the action log.
`BmtScriptPlayer -c <projectname> -a <actionlogname> -b2 -e20`	Create a project and apply the transactions numbered 2-20 from the action log.
`BmtScriptPlayer -m <projectname> -a <actionlogname> -e20`	Open an existing project and apply the transactions numbered 1-20 from the action log.
`BmtScriptPlayer-m <projectname> -a <actionlogname> -n`	Open an existing project and apply all the transactions from the action log. Do not save the project.

Fixing Errors Caused by Invalid Objects

You may encounter errors when running script files or verifying models if an object that is referenced by a transaction no longer exists, or if you renamed objects.

If an object no longer exists, retarget the missing object to another object.

Working with Scripts

If you are working with scripts and you retarget an object, all remaining script transactions use the new object. If the script stops for any other reason, you should modify the temporary project to correct the problem.

Note: Fixing errors by making changes to the main project can produce unpredictable results. Always fix errors by changing the temporary project.

When a script encounters errors, you can choose how you want to resolve the problem.

Solution	Action
Skip transactions that include this object	Click **Exclude** and in the **Exclude Transactions that Use this Object** dialog box, select the level of exclusion that you want. The current transaction and all subsequent ones that reference the excluded object are ignored. For example, if a transaction attempts to create a package that uses the excluded object, the package is not created. **Note:** We recommend that you attempt to fix errors before skipping transactions.
Replace this and all following occurrences of the object	Click **Replace** and in the **Replace Missing Objects** dialog box, select the option that you want.
Fix the problem manually	Click **Stop** and then fix the problem in the temporary project.

Retargeting an Object

If a transaction refers to an object that no longer exists, the script stops and a dialog box appears with the name of the problematic object. You can retarget the object by clicking **Replace** and selecting a new object.

If a missing object appears in an expression, the script stops and a dialog box appears with the name of the problematic object. You must fix the problem manually by opening the expression that contains the missing object.

Fixing Other Errors Encountered by the Script

You must fix script errors by modifying the temporary project. Fixing errors by making changes to the main project can produce unpredictable results.

Tip: You can move or minimize the **Synchronize** dialog box to view and modify the project.

Synchronize Projects

You can use Framework Manager log files to synchronize your project. You may choose to synchronize your project if you

- updated metadata in a third party modeling tool.
- made changes to metadata using a multidimensional modeling tool.

When you synchronize your project, you create a new project by replaying from the log files, all the actions you made in the original project.

Special considerations should be taken before synchronizing projects that contain segmented models, linked models, or repository connections.

If your data source is a relational database, you can update only the query subjects and do not need to perform a full project synchronization. You must perform a project synchronization to synchronize changes made in a third party data source.

If you import a subset of a data source, any new objects that were added to the data source are not included when you synchronize. The action log recorded the importing of objects that you originally specified. When you synchronize, only the originally imported objects are re-imported.

You can use project synchronization to run the complete action history of the model and update the model's metadata. You can also save portions of the action log to a separate script file for later use, or you can save the entire action log to a script file if you want to build the same model in batch mode. If you encounter errors when trying to run an action log script, see "Fixing Errors Caused by Invalid Objects" (p. 305).

After synchronizing, you can choose to accept the new changes and create a new project, or return to the original project. If you accept the new changes, the original project is replaced.

Because every action that you made in your project is rerun, synchronization may take a long time.

If an object that is referenced by a transaction no longer exists, either because it was renamed or deleted, you will receive errors during the synchronization. For example, if you imported a table named Products and then renamed the table to New Products in your data source, you will receive an error when you synchronize the project. The synchronization cannot detect that the table named New Products was previously imported using a different name. You must manually retarget the object to complete the synchronization. For information about fixing synchronization errors, see "Fixing Errors Caused by Invalid Objects" (p. 305).

Note: Action logs from Cognos ReportNet are not supported in this release.

Before synchronizing a project, you should understand how synchronization impacts segmented (p. 307) and linked (p. 307) models. You should also ensure that data source connections have not changed and that data sources are online. You can check your connections by testing a few key query subjects.

Segmented Models

A segmented model should be synchronized only by synchronizing the main project. The results of synchronizing the entire project are written to the log file of the main project. The ability to synchronize individual segments is lost after the first synchronization of the main project.

If you are working in the main project and change a segment, the main log file is updated. If you are working in the segment and make changes, the segment log file is updated.

Synchronization commands do not necessarily run in the order they appear in the log files. This happens because it is possible to update segments concurrently and the action logs are replayed based on the time of the original action. Commands may appear to jump between log files, making it difficult to use debugging features such as single stepping.

Linked Models

Log files are contained in the project that is open and not in the model that is updated.

If you open a main project and make changes to a linked model, the actions are logged in the log file of the main project. If you then synchronize the linked model, the change is lost because it did not appear in the set of log files that were used in the synchronization.

Synchronization can be run only on the main project or a stand-alone segment. You cannot synchronize linked projects or segments in the main project. If the segments are updated by the linked project, the synchronization can produce unpredictable results in the main project.

We recommended that you do not use model synchronization in combination with linked projects.

Projects Connected to a Repository

We do not recommend synchronizing projects that are connected to repositories. If you do synchronize repository projects, you should consider the following.

- All repository information will be removed from the project after the synchronization is complete.

  You must manually remove the project from the repository and use Framework Manager add the project back to the repository. All revision history will be lost.

- Synchronization will not be successful if the project was not added to the repository in the correct way.

  If you plan to synchronize a segment belonging to a repository project, you must first create the segment and use the **Add Project to Repository** action on the **Repository** menu. You should not add the project to the repository using the **Create Project** dialog box.

- If you use the View History action to revert to a previous version, the log files for the revisions that were supposed to be reverted are left intact. Without intervention, synchronization will execute these log files and will not correctly produce the intended revision. To successfully complete the synchronization, you will have to manually identify and remove the log files.

Steps to Synchronize

1. From the **Project** menu, click **Synchronize**.
2. We recommend that you create a backup of your Framework Manager project by selecting the **Backup project into this directory** check box.
3. If you want to view the details of a transaction, click the transaction.
4. Set the starting or stop point that you want.
   - To set the starting point for running the script, select the script and then click **Set the starting point**. You can do this at any time to skip an instruction or run instructions that have already been executed .
   - To set a stop point for the script, select the script and then click **Set the stop point** .

     You can stop the script to make a manual fix and then start it again.

     Tip: To remove the stop point, click **Remove the stop point**

     .

5. Using the toolbar buttons, choose the run action that you want.

Button	Description
	Runs the script After an error is encountered, clicking this button attempts to re-execute the failed instruction.
	Skips to the next transaction and runs the script to the end
	Runs the selected transaction only
	Skips to the next transaction and stops, but does not run any transactions

   The project window is updated as the script is run.

6. Fix any errors encountered by the script either by retargeting objects or modifying the temporary project as required.

   For more information, see "Fixing Errors Caused by Invalid Objects" (p. 305).

7. When the script has completed, click **Accept**.

   The original project is replaced by the contents of the temporary project.

   Tip: To return the project to its previous state, click **Revert**.

Query Behavior

By monitoring and adjusting the behavior of queries in your project, you can improve the performance of your model.

You can do the following:

- Set governors to reduce system resource requirements and improve performance by ensuring that the metadata in a package contains the appropriate limits. For example, you can set limits on the amount of data retrieved or the time that a query can take (p. 310).
- Specify whether aggregate rollups are computed locally or in the database. For relational metadata, you can improve performance by selecting the right type of query processing. After initial report execution, by turning the query reuse feature on, you can create reports without querying the database again (p. 314).
- Improve performance by setting the query processing type to determine whether SQL processing is performed by the database server or processed locally (p. 315).
- Improve performance by reusing cached data when running a report. By reusing cached data, you can create a report without querying the database again (p. 315).
- Select the vendor-specific function sets for the data sources defined in the project (p. 317).

- Indicate the behavior of individual functions based on the data sources in the project (p. 317).
- Control and optimize how queries are run by modifying the properties of a data source that was created using the Metadata Wizard in Framework Manager (p. 319).

Set Governors

Use governors to reduce system resource requirements and improve performance. You set governors before you create packages to ensure the metadata in the package contains the specified limits. All packages that are subsequently published use the new settings.

In a new project the governors do not have values defined in the model. You must open the **Governors** window and change the settings if necessary. When you save the values in the **Governors** window by clicking **OK**, the values for the governors are set. You can also set governors in Report Studio. The governor settings in Report Studio override the governor settings in the model.

Maximum Number of Report Tables

You can control the number of tables that a user can retrieve in a query or report. When a table is retrieved, it is counted each time it appears in the query or report. The limit is not the number of unique tables. If the query or report exceeds the limit set for the number of tables, an error message appears and the query or report is shown with no data.

A setting of zero (0) means no limit is set.

Maximum Number of Retrieved Rows

You can set data retrieval limits by controlling the number of rows that are returned in a query or report. Rows are counted as they are retrieved.

When you run a report and the data retrieval limit is exceeded, an error message appears and the query or report is shown with no data.

You can also use this governor to set limits to the data retrieved in a query subject test or the report design mode.

A setting of zero (0) means no limit is set.

If you externalize a query subject (p. 259), this setting is ignored when you publish the model.

Query Execution Time Limits

You can limit the time that a query can take. An error message appears when the preset number of seconds is reached.

A setting of zero (0) means no limit is set.

Large Text Item Limit

You can control the character length of BLOBS (binary large objects) that a user can retrieve in a query or report. When the character length of the BLOB exceeds the set limit, an error message appears, and the query or report is shown with no data.

A setting of zero (0) means no limit is set.

Outer Joins

You can control whether outer joins can be used in your query or report. An outer join retrieves all rows in one table, even if there is no matching row in another table. This type of join can produce very large, resource-intensive queries and reports.

Governors are set to deny outer joins by default. For example, outer joins are not automatically generated when you test a query item in Framework Manager.

SQL is generated automatically when you

- run a report
- test a query item or relationship in Framework Manager
- create a new model query subject based on other objects (p. 98)

If you keep the setting as **Deny**, you are notified only if you create a relationship in the Diagram tab that includes outer joins. You are not notified if you create a relationship in a data source query subject that includes outer joins.

If you set the governor to **Allow**, dimension to fact relationships are changed from inner joins to outer joins.

The outer joins governor does not apply in these circumstances:

- SQL that is generated by other means. If you set this governor to **Deny**, it does not apply to the permanent SQL found in a data source query subject, whether the SQL was generated on import (p. 65), manually entered, or based on existing objects (p. 87).
- Framework Manager needs to generate an outer join to create a stitched query. A stitched query is a query that locally combines the results of two or more sub-queries by using a locally processed outer join.

Note: This governor is not applicable for SAP BW data sources.

Cross-Product Joins

You can control whether cross-product joins can be used in your query or report. A cross-product join retrieves data from tables without joins. This type of join can take a long time to retrieve data.

The default value for this governor is **Deny**. Select **Allow** to allow cross-product joins.

Shortcut Processing

You can control how shortcuts are processed by Cognos 8.

When you open a model from a previous release, the **Shortcut Processing** governor is set to **Automatic**. **Automatic** is a shortcut that exists in the same folder as its target and behaves as an alias, or independent instance. However, a shortcut existing elsewhere in the model behaves as a reference to the original. When you create a new model, the **Shortcut Processing** governor is always set to **Explicit**.

If you set the governor to **Explicit**, the shortcut behavior is taken from the **Treat As** property. If the **Shortcut Processing** governor is set to **Automatic**, we recommend that you verify the model and,

when repairing, change the governor to **Explicit**. This changes all shortcuts to the correct value from the **Treat As** property based on the rules followed by the **Automatic** setting.

The **Shortcut Processing** governor takes priority over the **Treat As** property. For example, if the governor is set to **Automatic**, the behavior of the shortcut is determined by the location of the shortcut relative to its target regardless of the setting of the **Treat As** property is.

SQL Join Syntax

You can control how SQL is generated for inner joins in a model by selecting one of the following settings:

- The **Server determined** setting uses the `UseSQL99JoinSyntax` entry in the CQEConfig.xml file to determine the syntax. If there is not a `UseSQL99JoinSyntax` entry in the CQEConfig.xml file, then the **Implicit** setting is used. **Server determined** is the default value for this governor.
- The **Implicit** setting uses the `where` clause.

  For example,

  ```
  SELECT publishers.name, publishers.id,
  books.title FROM publishers, books WHERE publishers.id
  = books.publisher_id ORDER BY publishers.name, books.title;
  ```

- The **Explicit** setting uses the `from` clause with the keywords `inner join` in an `on` predicate.

  For example,

  ```
  SELECT
  publishers.name, publishers.id,
  books.title FROM publishers INNER JOIN books ON publishers.id
  = books.publisher_id ORDER BY publishers.name, books.title;
  ```

You can set the join type on the query property in Report Studio to override the value of this governor.

Regardless of the setting you use for this governor, the **Explicit** setting is used for left outer joins, right outer joins, and full outer joins.

This governor has no impact on typed-in SQL.

Allow Enhanced Model Portability at Run Time

This governor is selected upon initial upgrade of a Cognos ReportNet 1.x model. It prevents rigid enforcement of data types so that a Cognos 8 model can function as a ReportNet 1.x model until you update the data types in the metadata. After you have verified that the model has been upgraded successfully, clear this governor.

Other than for initial upgrade, there are limited uses for this governor. For example, you have created a model for use with a data source and you want to run it against a different data source. The new data source must be structurally similar to the original data source, and the database schema must be the same between the two data sources. If you select this governor, Cognos 8 retrieves metadata from the data source and caches it instead of using the metadata already cached in the model. When you have completed modifying and testing the model against the new data source, clear this governor.

If you do not use this governor, you must ensure that the following metadata is the same in the original and new data sources:

- collation sequence name
- collation level
- character set
- nullability
- precision
- scale
- column length
- data type

Allow Usage of Local Cache

Select this governor to specify that all reports based on this model will use cached data. For a new model, this governor is enabled by default.

This setting affects all reports that use the model. Use Report Studio if you want a report to use a different setting than the model. For more information, see "Improving Performance by Reusing Cached Data When Running a Report" (p. 315).

Allow Dynamic Generation of Dimension Information

This governor is selected only upon initial upgrade of a ReportNet 1.x model. This governor allows consistent behavior with ReportNet 1.x by deriving a form of dimension information from the relationships, key information, and index information in the data source.

Use With Clause When Generating SQL

You can choose to use the `With` clause with Cognos SQL if your data source supports the `With` clause.

The `With` clause is turned on for models created in Cognos 8. For upgraded models, it is turned off unless it was explicitly turned on in the ReportNet model prior to upgrading.

Suppress Null Values for SAP BW Data Sources

This governor is supported for SAP BW data sources only. It does not work with relational or OLAP data sources.

Some queries can be very large because null values are not filtered out. Null suppression removes a row or column for which all of the values in the row or column are null (empty). Null suppression is performed by SAP BW. This reduces the amount of data transferred to the Cognos 8 client products and improves performance.

The default value for this governor is to suppress null values. If you clear this governor, null values are not suppressed for any report or analysis in Cognos 8.

There is a property called **Suppress** in Report Studio that overrides this governor. If the **Suppress** property is set to **None,** null values are included in the result set even if the governor is set to suppress null values.

Steps

1. From the **Project** menu, click **Edit Governors**.
2. Specify the limits that you want to use when retrieving data.
3. Click **OK**.

Specify Where Aggregate Rollups are Processed

The Rollup Processing property for data sources determines whether aggregate rollups above the lowest level are computed locally or in the database. The default is set to local if local query processing is enabled, and is otherwise set to database.

Note: This property is not applicable for SAP BW data sources.

The possible options for this property are

- unspecified

  The aggregation rollup is not specified.

- local

  All aggregation rollups are computed locally (in the report server) using a running aggregate (for example, RSUM). Running aggregates spread the cost of this computation as the data is retrieved. Use this option if the local computer has more idle resources than the database computer, or if you find through experiment that it is the fastest method.

- database

  Aggregation rollups are computed by the underlying database software if possible. Otherwise, they are computed locally (provided local query processing is enabled). Running aggregates are used, but the cost is incurred by the database server instead of the report server. Use this option if the database computer has more idle resources than the local computer, or if you find through experiment that it is the fastest method.

- extended

  All aggregation rollups are computed by the database server using an extended aggregate (for example, XSUM). Extended aggregates incur the entire cost of this computation up front. Typically, this is the fastest method, but only where the database is set up to take advantage of materialized views. For databases where OLAP functionality is supported, this is translated into the appropriate OLAP aggregate functions.

Steps

1. In the **Project Viewer**, click the data source you want to change.

2. In the **Properties** pane, in the **Rollup Processing** list box, select the type of rollup processing that you want.

Improve Performance by Setting Query Processing Type

The query processing property for data sources determines whether SQL processing is performed by the database server or if it is processed locally. For relational metadata, you can improve performance by selecting the right type of query processing.

There are two types of query processing:

- limited local

  The database server does as much of the SQL processing and execution as possible. However, some reports or report sections use local SQL processing.

- database only

  The database server does all the SQL processing and execution. An error appears if any reports or report sections require local SQL processing.

Although the database server can usually run the SQL and run reports much faster, local processing is sometimes necessary. For example, choose limited local processing if you want to create cross database joins, or if you want your users to use unsupported SQL99 functions.

Some complex queries require limited local processing, such as a query that must generate an At clause to avoid double-counting. In this case, the query automatically uses limited local processing even if the package was published with database only processing.

For information about optimizing SAP BW performance, see "Optimizing SAP BW Performance" (p. 243).

Steps

1. In the **Project Viewer**, click the data source you want to change.
2. In the **Properties** pane, in the **Query Processing** list box, click either **Limited Local** or **Database Only**.

Improving Performance by Reusing Cached Data When Running a Report

When you run a report, the query request is sent to the database and the result set is returned. After the initial report execution, you may decide to make changes to the report. Often, the report can be created without querying the database again. To take advantage of this, turn the query reuse feature on.

When query reuse is turned on and you run a report for the first time, the query is stored in the cache on the report server. Also, some data source resources may not be available until the transaction using them is released. The current default time-out is 60 minutes. If certain database activities involve modifying database objects, you must wait for the time-out period to be completed, or you can disable query reuse for reports.

The first time the report is run and the cache is created, the response time may be slightly negatively impacted. The performance improvement is realized by the report consumer on each subsequent report execution, when the response time is improved by as much as 80%. This performance improvement occurs because the report does not have to re-query the database. In addition to this, reduced queries to the database yields improved overall system performance, which positively impacts all users.

Query reuse can be set on the model or on individual reports. To specify that all reports using a particular model should use cached data, enable the **Allow Usage of Local Cache** governor on the model in Framework Manager and republish the model. By default, this setting affects all reports that use this model, including analyses that are run as reports in Cognos Viewer.

Query Reuse in Cognos Viewer

If you want a report to use a different setting than the model, you can do this in Report Studio. In the **Properties** pane, change the **Use Local Cache** property. Set the property to **No** if you want to always execute the query. Set the property to **Yes** if you want to use cached results. If you want the report to use the same setting as the model, change the setting to **Default**.

Changing the **Use Local Cache** property for one report does not affect other reports.

Reusing Cached Data in Query Studio

Query Studio reuses cached data under various conditions. If query reuse is turned on in the model and the action can be satisfied by a subset of the cached data set, the report uses the cached data. For example, changes to the report such as adding a filter or removing a column, may change the report data, but the request can still be satisfied from a subset of the cached data.

If query reuse is turned off and the action can be satisfied from the cached data set without modifications, the report still uses the cached data. For example, changing the report format uses the previous data set even if query reuse is turned off. This is known as cursor reuse. Cursor reuse is used when the cached data can satisfy the request without modifications.

Reports that were created in Query Studio always use the same setting as that specified in the model. If the model has query reuse turned on, the report attempts to use the cached data.

Deciding Whether to Use Query Reuse in Your Environment

Before deciding whether or not to turn query reuse on, consider the following:

- If most report consumers run reports interactively but run them only once, you may not experience a high level of performance improvement by caching data.

  Note: Regardless of the query reuse settings, reports that run in batch mode do not cache data.

- The size of the cache may impact Cognos 8 scalability. For example, if a report has a large result set, the cache will also be large. This should be taken into account when sizing and configuring your server environment.

Select Function Sets

A collection of vendor-specific functions is called a function set. When you create a project that contains relational metadata, the expression editor lists the function sets for all available vendors. However, you can restrict the function sets so that they list only the vendors that you want to use in your project. You customize the function set by identifying the specific vendor for each data source defined in the project.

You can use functions that you defined in your relational data source in Framework Manager. If you imported the user-defined functions, they are listed in Framework Manager for easy selection. If you did not import them, you can type the name of the function into an expression. If the function must be qualified, you must import them into Framework Manager.

Sometimes the vendor-specific functions were created on schemas with broad access permissions. You cannot use these functions in Cognos 8 on a schema with restricted access permissions if both the schemas are on the same database instance.

If an unrecognized function is typed into a report, it is assumed that the function is native. For more information, see "Native SQL" (p. 112).

Note: When you create a project that contains SAP BW metadata, Framework Manager automatically lists only the functions that apply to SAP BW data.

Steps

1. From the **Project** menu, click **Project Function List**.
2. Select the **Set function list based on the data source type** check box.

   Tip: To disable this filter, select the **Include all function sets** check box.
3. In the **Function set** page, click the appropriate data source row.
4. Select the function set you want to use with this data source.
5. Repeat steps 2 to 4 until finished.
6. Click **OK**.

For information about how to specify which function sets are included in individual packages, see "Optimizing SAP BW Performance" (p. 243).

Quality of Service

With Framework Manager, you can query any combination of data source types, but not all data sources support functions the same way. The quality of service indicator provides you and your users with a visual clue about the behavior of individual functions when used in conjunction with the data sources in the model.

Each function specified in your data source may have a different quality of service, depending on the type of data source in use. For each query feature that does not have the same quality of service across packages, you can override the level of service and add text to describe the specific situation

in that model. Your users can see the quality of service indicators and the context specific description, and use this information when determining which functions to use in reports.

The quality of service for a function is specified at the data source level and can be set for an individual function (p. 387) or for all functions in a package. The quality of service indicators are:

- not available (X)

  This function is not available for any data sources in the package

- limited availability (!!)

  The function is not available for some data sources in the package

- poor performance (!)

  The function is available for all data sources in the package but may have poor performance in some data sources

- unconstrained (no symbol)

  The function is available for all data sources

If there is more than one type of data source in the model, the quality of service values are aggregated according to the following rules:

- If the quality of service is defined as Unconstrained, Poor Performance, or Limited Availability in one data source and defined as Not Available in another data source, the quality of service for that function becomes Limited Availability.

- In all other cases, the lowest common dominator is used. For example, if the quality of service is Unconstrained in one data source and Poor Performance in another data source, the quality of service for that function becomes Poor Performance. If the quality of service is Poor Performance in one data source and Limited Availability in another data source, the quality of service is reported as Limited Availability.

Impact of Overriding the Quality of Service Indicator

Framework Manager determines the quality of service for functions based upon the data source type. Taking into consideration the context of the model, you can override the quality of service that is determined by the product. Overriding the quality of service provides guidance to your users. It does not change the level of support for that function in your data source.

When a package is made by combining sub-packages, quality of service overrides in the parent package take precedence. If there is no parent override, the quality of service for the child packages are aggregated as described above.

Consider Your Users

Ultimately, the goal is to provide your users with enough information to satisfy their business requirements, but not enough to confuse them. If your users are unable to make decisions regarding which functions to use based on the quality of service indicators, you should consider publishing separate packages for different groups of users. If your users require access to functions whose

quality of service is less than Unconstrained, you should document the restrictions of those functions when you set the quality of service.

Impacts on Performance

The quality of service indicators has no direct impact on query performance. Service indicators are intended to give you some control over which functions are available for use. You can then prevent your users from using functions that could result in long running queries or queries that fail.

It is important to note that if you use functions that are not available in your data source, Framework Manager tries to compensate by using local processing on the report server. This may have an impact on query performance because the work is done on your report server instead of on your data source server.

In some situations, local processing may require more data to be retrieved from the data source server, which has an impact on both the data source server and the network. For example, OLAP functions are not available in a relational data source. If you attempt to use OLAP functions with a relational data source, Framework Manager uses the dimensional information in the data source to generate a local cube and run the OLAP functions against the cube. This requires retrieval of dimensional information from the data source server and extra processing on the report server.

Steps

1. From the **Project** menu, select **Project Function List**.
2. Click **Define Quality of Service**.
3. Expand the tree nodes to view the quality of service for each function.
4. To override the quality of service, click the arrow beside each function and select the quality of service indicator from the list.
5. After changing the quality of service, you can add detailed information about the function in the text box on the right.

   This information becomes available to your users and can assist them in determining whether to use this function in their reports.

   Tip: Click **Remove override** to set the quality of service back to the default.
6. Click **OK**.

Control and Optimize How Queries Are Run

You can modify the properties of a data source that was created using the **Metadata Wizard** in Framework Manager. The data source properties help you control and optimize the way queries are run against the database.

You cannot modify the properties of a data source that was created using the portal. These data sources can only be modified in the portal. For more information, see the *Administration and Security Guide*.

Data Source Property	Description
Name	Descriptive name of the data source connection provided by the user at the time of creation
Query Processing	Determines whether SQL processing is performed by the database server or processed locally
Rollup Processing	Determines whether aggregate rollups are computed locally or in the database
Content Manager Data Source	Specifies the name of the data source as it is identified in the Content Manager. If using an XML data source, this property may be parameterized.
Catalog	Represents different information for different databases. For example, if the database is SQL Server, the element contains the name of the database; if the database is Oracle, it is not used.
Cube	Specifies the name of the cube
Schema	Represents different information for different databases. For example, for SQL Server or Oracle, the element contains the name of the owner.
Type	Specifies the type for the parent object.
Query Type	Specifies the type of query model that this data source understands. For example, SQL sources are relational and MDS sources are multidimensional.
Query Interface	This element contains two letters, identifying the provider type. It is maintained by the application.
Function Set ID	Defines the function set that applies to a data source. Used in the initial population of the function sets of a security view when a package is created.
Parameter Maps	References a parameterMap that represents a DB2OLAP alias table map.

Chapter 8: Guidelines for Modeling Metadata

Framework Manager is a metadata modeling tool that drives query generation for Cognos 8. A model is a collection of metadata that includes physical information and business information for one or more data sources. Cognos 8 enables performance management on normalized and denormalized relational data sources as well as a variety of OLAP data sources.

The first section of this document discusses fundamental Cognos 8 modeling concepts that you need to understand about modeling metadata for use in business reporting and analysis (p. 321).

The next section discusses building the relational model (p. 337).

This document is available in English, French, German, and Japanese. To access it in a different language, go to *installation location*\cognos\c8\webcontent\documentation and open the folder for the language you want. Then open ug_best.pdf.

Understanding Cognos 8 Modeling Concepts

Before you begin, there are concepts that you need to understand.

Relational modeling concepts:

- cardinality (p. 322)
- determinants (p. 324)
- multiple-fact, multiple-grain queries (p. 327)

Model design considerations:

- relationships and determinants (p. 330)
- minimized SQL (p. 331)
- metadata caching (p. 332)
- query subjects vs. dimensions (p. 333)
- shortcuts vs. copies of query subjects (p. 333)
- folders vs. namespaces (p. 335)

Dimensional modeling concepts:

- regular dimensions (p. 336)
- measure dimensions (p. 336)
- scope relationships (p. 336)

Relational Modeling Concepts

When modeling in Framework Manager, it is important to understand that there is no requirement to design your data source to be a perfect star schema. Snowflaked and other forms of normalized schemas are equally acceptable as long as your data source is optimized to deliver the performance you require for your application. In general, we recommend that you create a logical model that conforms to star schema concepts. This is a requirement for Analysis Studio and has also proved to be an effective way to organize data for your users.

When beginning to develop your application with a complex data source, it is recommended that you create a simplified view that represents how your users view the business and that is designed using the guidelines in this document to deliver predictable queries and results. A well-built relational model acts as the foundation of your application and provides you with a solid starting point if you choose to take advantage of dimensional capabilities in Cognos 8.

If you are starting with a star schema data source, less effort is required to model because the concepts employed in creating a star schema lend themselves well to building applications for query and analysis. The guidelines in this document will assist you in designing a model that will meet the needs of your application.

Cardinality

Relationships exist between two query subjects. The cardinality of a relationship is the number of related rows for each of the two query subjects. The rows are related by the expression of the relationship; this expression usually refers to the primary and foreign keys of the underlying table.

Cognos 8 uses the cardinality of a relationship in the following ways:

- to avoid double-counting fact data
- to support loop joins that are common in star schema models
- to optimize access to the underlying data source system
- to identify query subjects that behave as facts or dimensions

A query that uses multiple facts from different underlying tables is split into separate queries for each underlying fact table. Each single fact query refers to its respective fact table as well as to the dimensional tables related to that fact table. Another query is used to merge these individual queries into one result set. This latter operation is generally referred to as a stitched query. You know that you have a stitched query when you see `coalesce` and a full outer join.

A stitched query also allows Cognos 8 to properly relate data at different levels of granularity (p. 327).

Cardinality in Generated Queries

Framework Manager supports both minimum-maximum cardinality and optional cardinality.

In `0:1`, `0` is the minimum cardinality, `1` is the maximum cardinality.

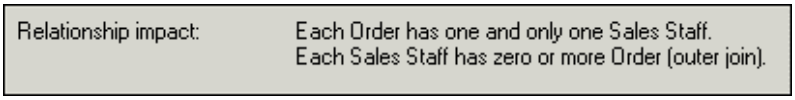
In `1:n` , `1` is the minimum cardinality, `n` is the maximum cardinality.

A relationship with cardinality specified as 1:1 to 1:n is commonly referred to as 1 to n when focusing on the maximum cardinalities.

A minimum cardinality of 0 indicates that the relationship is optional. You specify a minimum cardinality of 0 if you want the query to retain the information on the other side of the relationship in the absence of a match. For example, a relationship between customer and actual sales may be specified as 1:1 to 0:n. This indicates that reports will show the requested customer information even though there may not be any sales data present.

Therefore a 1 to n relationship can also be specified as 0:1 to 0:n, 0:1 to 1:n, or 1:1 to 0:n.

Use the **Relationship impact** statement in the **Relationship Definition** dialog box to help you understand cardinality. For example, Sales Staff (1:1) is joined to Orders (0:n).



It is important to ensure that the cardinality is correctly captured in the model because it determines the detection of fact query subjects and it is used to avoid double-counting factual data.

When generating queries, Cognos 8 follows these basic rules to apply cardinality:

- Cardinality is applied in the context of a query.
- 1:n cardinality implies fact data on the n side and implies dimension data on the 1 side.
- A query subject may behave as a fact query subject or as a dimensional query subject, depending on the relationships that are required to answer a particular query.

Use the **Model Advisor** to see an assessment of the behavior implied by cardinality in your model.

For more information, see "Single Fact Query" (p. 351) and "Multiple-fact, Multiple-grain Query on Conformed Dimensions " (p. 352).

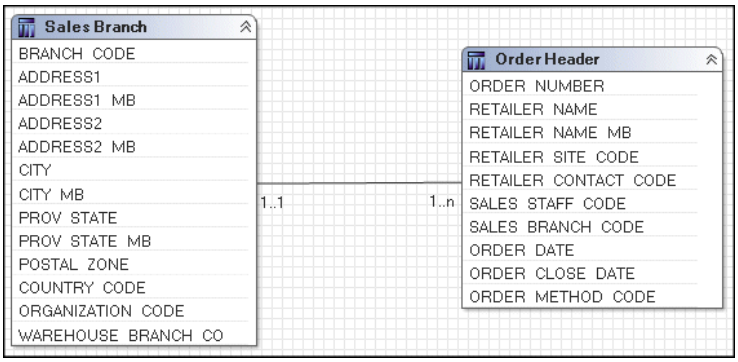
Cardinality in the Context of a Query

The role of cardinality in the context of a query is important because cardinality is used to determine when and where to split the query when generating multiple-fact queries. If dimensions and facts are incorrectly identified, stitched queries can be created unnecessarily, which is costly to performance, or the queries can be incorrectly formed, which can give incorrect results (p. 362).

The following examples show how cardinality is interpreted by Cognos 8.

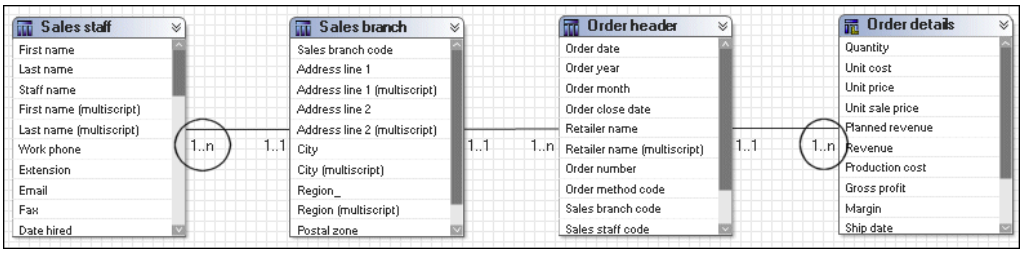
Example: Query Subjects Behaving as a Dimension and a Fact

In this example, Sales Branch behaves as a dimension relative to Order Header and Order Header behaves as a fact relative to Sales Branch.



Example: Four Query Subjects Included in a Query

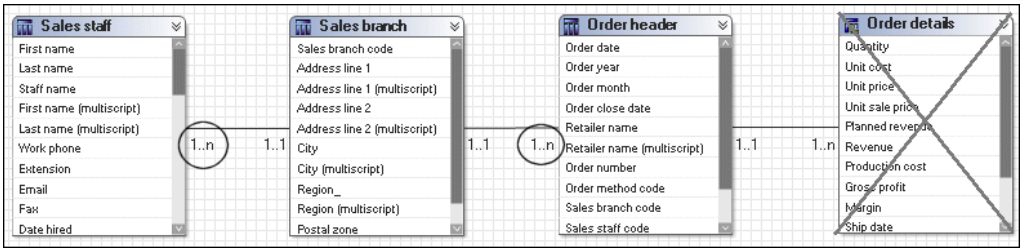
In this example, all four query subjects are included in a query. The diagram shows that Sales staff and Order details are treated as facts. Order header and Sales branch are treated as dimensions.



The SQL generated for this query will be split, treating Sales staff and Order details as facts and getting incorrect results. The results of these two subqueries are stitched using the information retrieved from Sales branch. This gives a report that lists the Sales staff information by Sales branch next to the Order details and Order header information by Sales branch. For more information, see "Resolving Queries That Are Split in the Wrong Place " (p. 365).

Example: Three Query Subjects Included in a Query

In this example, only three query subjects are included in a query. Order details is not used. Order header is now treated as a fact. Sales staff continues to be treated as a fact.



The SQL in this example also generates a stitched query, which returns a similar result as above. Note that a stitch operation retains the information from both sides of the operation by using a full outer join.

Determinants

Determinants reflect granularity by representing subsets or groups of data in a query subject and are used to ensure correct aggregation of this repeated data. Determinants are most closely related to the concept of keys and indexes in the data source and are imported based on unique key and

index information in the data source. We recommend that you always review the determinants that are imported and, if necessary, modify them or create additional ones. By modifying determinants, you can override the index and key information in your data source, replacing it with information that is better aligned with your reporting and analysis needs. By adding determinants, you can represent groups of repeated data that are relevant for your application.

An example of a unique determinant is Day in the Time example below. An example of a non-unique determinant is Month; the key in Month is repeated for the number of days in a particular month. When you define a non-unique determinant, you should specify group by. This indicates to Cognos 8 that when the keys or attributes associated with that determinant are repeated in the data, it should apply aggregate functions and grouping to avoid double-counting. It is not recommended that you specify determinants that have both uniquely identified and group by selected or have neither selected.

Year Key	Month Key	Month Name	Day Key	Day Name
2006	200601	January 06	20060101	Sunday, January 1, 2006
2006	200601	January 06	20060102	Monday, January 2, 2006

You can define three determinants for this data set as follows -- two group by determinants (Year and Month) and one unique determinant (Day). The concept is similar but not identical to the concept of levels and hierarchies.

Name of the Determinant	Key	Attributes	Uniquely Identified	Group By
Year	Year Key	None	No	Yes
Month	Month Key	Month Name	No	Yes
Day	Day Key	Day Name Month Key Month Name Year Key	Yes	No

In this case, we use only one key for each determinant because each key contains enough information to identify a group within the data. Often Month is a challenge if the key does not contain enough information to clarify which year the month belongs to. In this case, however, the Month key includes the Year key and so, by itself, is enough to identify months as a sub-grouping of years.

Note: While you can create a determinant that groups months without the context of years, this is a less common choice for reporting because all data for February of any year would be grouped together instead of all data for February 2006 being grouped together.

Using Determinants with Multiple-Part Keys

In the Time dimension example above, one key was sufficient to identify each set of data for a determinant but that is not always the case.

For example, the following Geography dimension uses a combination of unique and group by foreign keys. This multiple-part key defines a set in the data.

Region	Country Key	State/Province Key	City Key
North America	USA	Illinois	Springfield
North America	USA	Missouri	Springfield
North America	USA	California	Dublin
Europe	Ireland	n/a	Dublin

Similar to the example about Time, you can define three determinants for this data set as follows -- two group by determinants (Country and State/Province) and one unique determinant (City). The difference is how you define the key for the City determinant to make it unique.

Name of the Determinant	Key	Attributes	Uniquely Identified	Group By
Country	Country Key	None	No	Yes
State/Province	State/Province Key	None	No	Yes
City	Country Key State/Province Key City Key	None	Yes	No

In this case, we used both Country Key and State/Province Key to ensure uniqueness for City. We did this because in the data we were given, some city names were repeated across countries that did not have data for the State/Province Key.

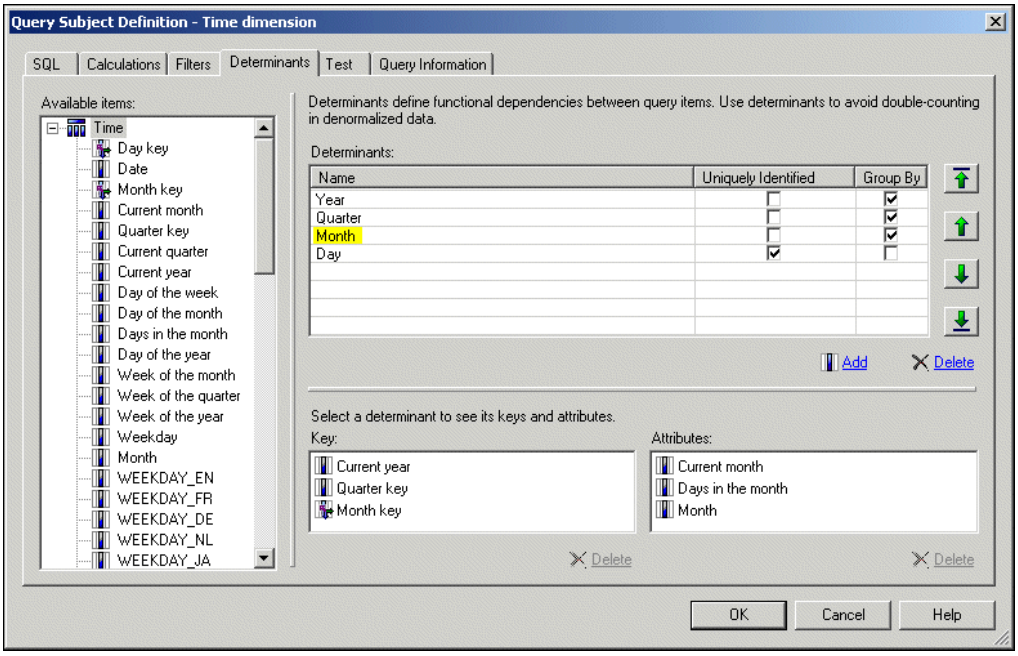
Determinants Are Evaluated in the Order In Which They Are Specified

There is no concept of a hierarchy in determinants, but there is an order of evaluation. When Cognos 8 looks at a selection of items from a query subject, it compares them to each determinant (keys and attributes) one at a time in the order that is set in the **Determinants** tab. In this way, Cognos 8 selects the determinant that is the best match.

In the following example, the attributes current month, days in month, and localized month names are associated to the Month key. When a query is submitted that references any one of these attributes, the Month determinant is the first determinant on which the matching criteria is satisfied.

If no other attributes are required, the evaluation of determinants stops at Month and this determinant is used for the `group` and `for` clauses in the SQL.

In cases where other attributes of the dimension are also included, if those attributes have not been matched to a previous determinant, Cognos 8 continues evaluating until it finds a match or reaches the last determinant. It is for this reason that a unique determinant has all query items associated to it. If no other match is found, the unique key of the entire data set is used to determine how the data is grouped.



When to Use Determinants

While determinants can be used to solve a variety of problems related to data granularity, you should always use them in the following primary cases:

- A query subject that behaves as a dimension has multiple levels of granularity and will be joined on different sets of keys to fact data.

  For example, Time has multiple levels, and it is joined to Inventory on the Month Key and to Sales on the Day Key. For more information, see "Multiple-fact, Multiple-grain Queries" (p. 327).

- There is a need to count or perform other aggregate functions on a key or attribute that is repeated.

  For example, Time has a Month Key and an attribute, Days in the month, that is repeated for each day. If you want to use Days in the month in a report, you do not want the sum of Days in the month for each day in the month. Instead, you want the unique value of Days in the month for the chosen Month Key. In SQL, that is `XMIN(Days in the month for Month_Key)` . There is also a `Group by` clause in the Cognos SQL.

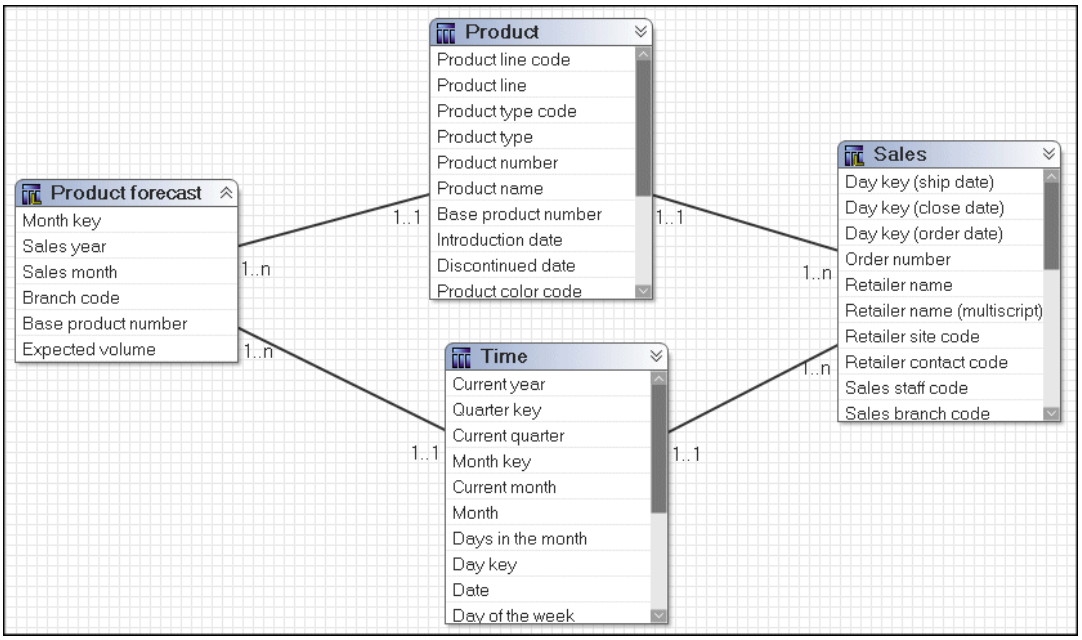
Multiple-fact, Multiple-grain Queries

Note that in this section, the term dimension is used in the conceptual sense. A query subject with cardinality of `1:1` or `0:1` behaves as a dimension. For more information, see "Cardinality" (p. 322).

Multiple-fact, multiple-grain queries in relational data sources occur when a table containing dimensional data is joined to multiple fact tables on different keys. A dimensional query subject typically has distinct groups, or levels, of attribute data with keys that repeat. The Cognos 8 studios automatically aggregate to the lowest common level of granularity present in the report. The potential for double-counting arises when creating totals on columns that contain repeated data. When the level of granularity of the data is modeled correctly, double-counting can be avoided.

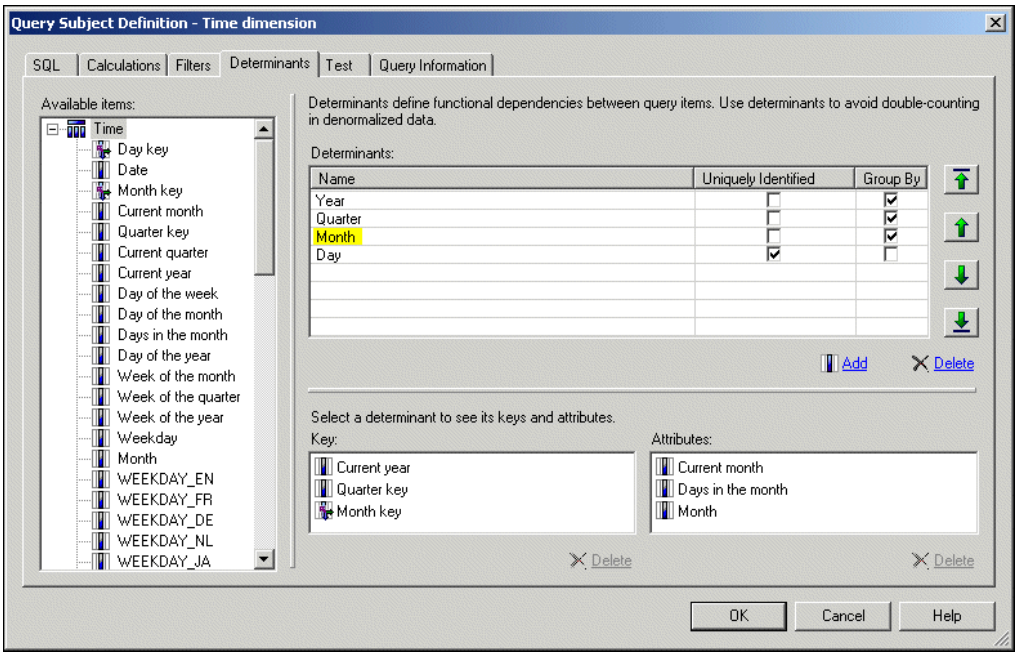
Note: You can report data at a level of granularity below the lowest common level. This causes the data of higher granularity to repeat, but the totals will not be affected if determinants are correctly applied.

This example shows two fact query subjects, Sales and Product forecast, that share two dimensional query subjects, Time and Product.



Time is the focal point of the granularity issue in this example. Sales is joined to Time on the Day key, and Product forecast is joined to Time on the Month key. Because of the different join keys, a minimum of two determinants must be clearly identified on Time. For example, the determinants for Month and Day have their keys identified. Day is the unique key for Time, Month keys are repeated for each day in the month.

For example, the determinants for Month are as follows.



The Product query subject could have at least three determinants: Product line, Product type, and Product. It has relationships to both fact tables on the Product key. All joins occur on the Product key so there are no granularity issues.

By default, a report is aggregated to retrieve records from each fact table at the lowest common level of granularity. If you create a report that uses Quantity from Sales, Expected volume from Product forecast, Month from Time, and Product name from Product, the report retrieves records from each fact table at the lowest common level of granularity. In this example, it is at the month and product level.

To prevent double-counting when data exists at multiple levels of granularity, create at least two determinants for the Time query subject. For an example, see "Determinants" (p. 324).

Month	Product name	Quantity	Expected volume
April 2007	Aloe Relief	1,410	1,690
April 2007	Course Pro Umbrella	132	125
February 2007	Aloe Relief	270	245
February 2007	Course Pro Umbrella		1
February 2006	Aloe Relief	88	92

If you do not specify the determinants properly in the Time query subject, incorrect aggregation may occur. For example, Expected volume values that exist at the Month level in Product forecast is repeated for each day in the Time query subject. If determinants are not set correctly, the values for Expected volume are multiplied by the number of days in the month.

Month	Product name	Quantity	Expected volume
April 2007	Aloe Relief	1,410	50,700
April 2007	Course Pro Umbrella	132	3,750
February 2007	Aloe Relief	270	7,134
February 2007	Course Pro Umbrella		29
February 2006	Aloe Relief	88	2,576

Note the different numbers in the Expected volume column.

Model Design Considerations

When building a model, it is important to understand that there is no single workflow that will deliver a model suitable for all applications. Before beginning your model, it is important to understand the application requirements for functionality, ease of use, and performance. The design of the data source and application requirements will determine the answer to many of the questions posed in this section.

Where Should You Create Relationships and Determinants?

A frequently asked question is where to create relationships. Should relationships be created between data source query subjects, between model query subjects, or between both? The answer may vary because it depends on the complexity of the data source that you are modeling.

When working with data source query subjects, relationships and determinants belong together.

When working with model query subjects, there are side effects to using relationships and determinants that you should consider:

- The model query subject starts to function as a view, which overrides the **As View** or **Minimized** setting in the **SQL Generation** type for a query subject.

  This means that the SQL stays the same no matter which items in the query subject are referenced. For more information, see "What Is Minimized SQL?" (p. 331).

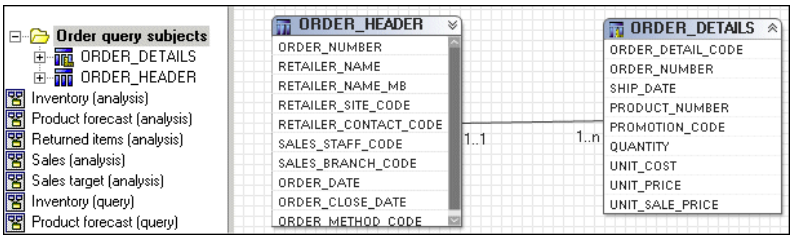
- The model query subject becomes a stand-alone object.

  This means underlying relationships are no longer applied, except those between the objects that are referenced. It may be necessary to create additional relationships that were previously inferred from the metadata of the underlying query subjects.

- When a determinant is created on a model query subject, the determinant is ignored unless a relationship is also created.

Here is an example of a relationship on a model query subject that purposely overrides the **Minimized SQL** setting and simplifies the model. In the GO Sales sample model, Order Header and Order

Details are combined so that they behave as a single fact. They are placed in their own folder and all relationships to them are deleted except the relationship between Order Header and Order Details. This is the only relationship that will matter after a model query subject is created and relationships attached to it.



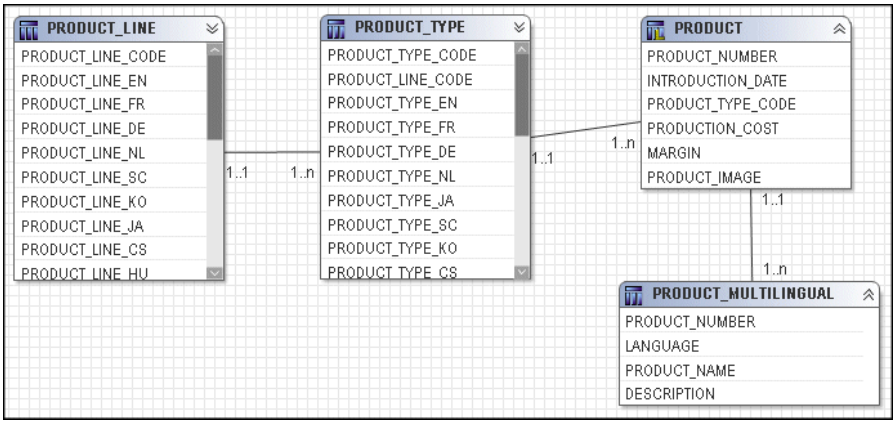
To decide where to specify relationships and determinants in the model, you must understand the impact of minimized SQL to your application.

For more information about relationships, determinants, and minimized SQL, see the **Model Advisor** topics in the Framework Manager *User Guide*.

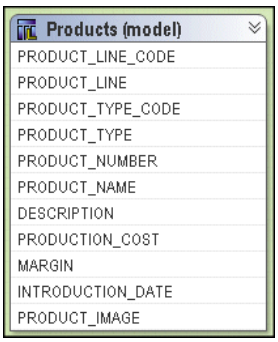
What Is Minimized SQL?

When you use minimized SQL, the generated SQL contains only the minimal set of tables and joins needed to obtain values for the selected query items.

To see an example of what minimized SQL means, you can use the Product tables in the GO Sales sample model. Four query subjects, Product Line, Product Type, Product, and Product Multilingual all join to each other.



They can be combined in a model query subject.



If you test the Products model query subject as a whole, you see that four tables are referenced in the `from` clause of the query.

```
Cognos SQL
select
       PRODUCT_LINE.PRODUCT_LINE_CODE  as  Product_Line_Code,
       PRODUCT_LINE.PRODUCT_LINE_EN  as  Product_Line,
       PRODUCT_TYPE.PRODUCT_TYPE_CODE  as  Product_Type_Code,
       PRODUCT_TYPE.PRODUCT_TYPE_EN  as  Product_Type,
       PRODUCT.PRODUCT_NUMBER  as  Product_Number,
       PRODUCT_MULTILINGUAL.PRODUCT_NAME  as  Product_Name,
       PRODUCT_MULTILINGUAL.DESCRIPTION  as  Product_Description,
       PRODUCT.INTRODUCTION_DATE  as  Introduction_Date,
       PRODUCT.PRODUCT_IMAGE  as  Product_Image,
       PRODUCT.PRODUCTION_COST  as  Production_Cost,
       PRODUCT.MARGIN  as  Margin
from
       gosl_82..gosl.PRODUCT_LINE PRODUCT_LINE,
       gosl_82..gosl.PRODUCT_TYPE PRODUCT_TYPE,
       gosl_82..gosl.PRODUCT PRODUCT,
       gosl_82..gosl.PRODUCT_MULTILINGUAL PRODUCT_MULTILINGUAL
where
       (PRODUCT_MULTILINGUAL."LANGUAGE" = N'EN') and
       (PRODUCT_LINE.PRODUCT_LINE_CODE = PRODUCT_TYPE.PRODUCT_LINE_CODE) and
       (PRODUCT_TYPE.PRODUCT_TYPE_CODE = PRODUCT.PRODUCT_TYPE_CODE) and
       (PRODUCT.PRODUCT_NUMBER = PRODUCT_MULTILINGUAL.PRODUCT_NUMBER)
```

If you test only Product name, you see that the resulting query uses only Product Multilingual, which is the table that was required. This is the effect of minimized SQL.

```
Cognos SQL
select
       PRODUCT_MULTILINGUAL.PRODUCT_NAME  as  Product_Name
from
       gosl_82..gosl.PRODUCT_MULTILINGUAL PRODUCT_MULTILINGUAL
where
       (PRODUCT_MULTILINGUAL."LANGUAGE" = N'EN')
```

Example: When Minimized SQL Is Important

If you are modeling a normalized data source, you may be more concerned about minimized SQL because it will reduce the number of tables used in some requests and perform better. In this case, it would be best to create relationships and determinants between the data source query subjects and then create model query subjects that do not have relationships.

There is a common misconception that if you do not have relationships between objects, you cannot create star schema groups. This is not the case. Select the model query subjects to include in the group and use the **Star Schema Grouping** wizard. Or you can create shortcuts and move them to a new namespace. There is no need to have shortcuts to the relationships; this feature is purely visual in the diagram. The effect on query generation and presentation in the studios is the same.

Example: When Minimized SQL Is Not as Important as Predictable Queries

There may be some elements in a data source that you need to encapsulate to ensure that they behave as if they were one data object. An example might be a security table that must always be joined to a fact. If you look at the GO Sales sample model, Order Header and Order Details are a set of tables that together represent a fact and you would always want them to be queried together. For an example, see "Where Should You Create Relationships and Determinants?" (p. 330).

What Is Metadata Caching?

Framework Manager stores the metadata that is imported from the data source. However depending on governor settings and certain actions you take in the model, this metadata might not be used when preparing a query. If you enable the **Allow enhanced model portability at run time** governor, Framework Manager always queries the data source for information about the metadata before preparing a query. If you have not enabled this governor, in most cases Framework Manager accesses

the metadata that has been stored in the model instead of querying the data source. The main exceptions are:

- The SQL in a data source query subject has been modified. This includes the use of macros.
- A calculation or filter has been added to a data source query subject.

Note: The metadata queries generated by Cognos 8 are well supported by most relational database management system vendors and should not have a noticeable impact on most reporting applications.

Query Subjects vs. Dimensions

Query subjects and dimensions serve separate purposes. The query subject is used to generate relational queries and may be created using star schema rules, while the dimension is used for dimensional modeling of relational sources, which introduces OLAP behavior. Because query subjects are the foundation of dimensions, a key success criterion for any dimensional model is a sound relational model.

A dimensional model is required only if you want to use Analysis Studio, to enable drilling up and down in reports, or to access member functions in the studios. For many applications, there is no need for OLAP functionality. For example, your application is primarily for ad hoc query or reporting with no requirement for drilling up and down. Or you are maintaining a Cognos ReportNet model. In these cases, you may choose to publish packages based on query subjects alone.

Determinants for query subjects are not the same as levels and hierarchies for regular dimensions but they can be closely related to a single hierarchy. If you are planning to use your query subjects as the foundation for dimensions, you should consider the structure of the hierarchies you expect to create and ensure that you have created determinants that will support correct results when aggregating. Ensure that you have the following:

- The query subject should have a determinant specified for each level of the hierarchy in the regular dimension.
- The determinants should be specified in the same order as the levels in the regular dimension.
- If you expect to have multiple hierarchies that aggregate differently, you may need to consider creating an additional query subject with different determinants as the source for the other hierarchy.

By creating a complete relational model that delivers correct results and good performance, you will have a strong foundation for developing a dimensional model. In addition, by ensuring that a layer of model objects, either query subjects or dimensions, exists between the data source and the objects exposed to the studios, you are better able to shield your users from change.

Model Objects vs. Shortcuts

The key difference between model objects and shortcuts is that model objects give you the freedom to include or exclude items and to rename them. You may choose to use model objects instead of shortcuts if you need to limit the query items included or to change the names of items.

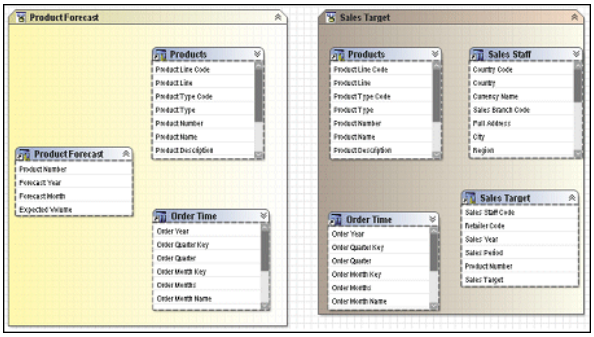
Shortcuts are less flexible from a presentation perspective than model objects, but they require much less maintenance because they are automatically updated when the target object is updated. If

maintenance is a key concern and there is no need to customize the appearance of the query subject, use shortcuts.

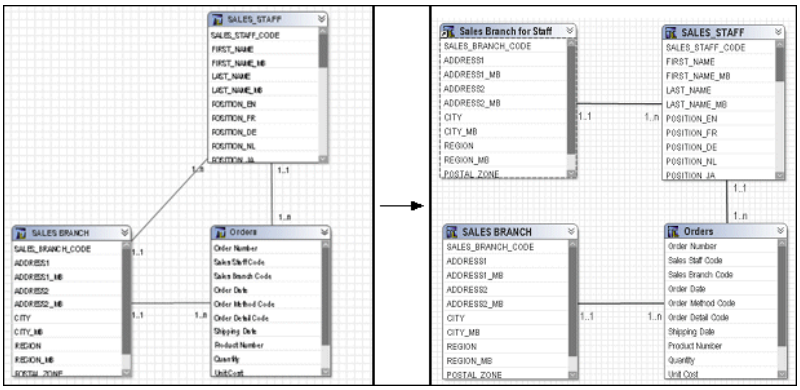
Framework Manager has two types of shortcuts:

- regular shortcuts, which are a simple reference to the target object.
- alias shortcuts, which behave as if they were a copy of the original object with completely independent behavior. Alias shortcuts are available only for query subjects and dimensions.

Regular shortcuts are typically used as conformed dimensions with star schema groups, creating multiple references with the exact same name and appearance in multiple places. In the example below, the shortcuts created for Products and Order Time behave as references. If a query is written that brings Products from both Product Forecast and Sales Target, the query uses the definition of Products based on the original and this definition appears only once in the query.



Alias shortcuts are typically used in role-playing dimensions or shared tables. Because there is already an example in this document for role-playing dimensions, we will look at the case of shared tables. In this example, Sales Staff and Sales Branch can be treated as different hierarchies. From our knowledge of the data, we know that because staff can move between branches, we need to be able to report orders against Sales Branch and Sales Staff independently as well as together. To achieve this, we need to create an alias to Sales Branch that can be used as a level in the Sales Staff hierarchy.



With the new alias shortcut in place, it is possible to create queries that require orders by sales branch and orders by sales staff with their current branch information simultaneously.

Being able to specify the behavior of shortcuts is new to Cognos 8.3.

When you open a model from a previous release, the **Shortcut Processing** governor is set to **Automatic**. When **Automatic** is used, shortcuts work the same as in previous releases, that is, a

shortcut that exists in the same folder as its target behaves as an alias, or independent instance, whereas a shortcut existing elsewhere in the model behaves as a reference to the original. To take advantage of the **Treat As** property, it is recommended that you verify the model and, when repairing, change the governor to **Explicit**. The repair operation changes all shortcuts to the correct value from the **Treat As** property based on the rules followed by the **Automatic** setting, this means that there should be no change in behavior of your model unless you choose to make one or more changes to the **Treat As** properties of your shortcuts.

When you create a new model, the **Shortcut Processing** governor is always set to **Explicit**.

When the governor is set to **Explicit**, the shortcut behavior is taken from the **Treat As** property and you have complete control over how shortcuts behave without being concerned about where in the model they are located.

Folders vs. Namespaces

The most important thing to know about namespaces is that once you have begun authoring reports, any changes you make to the names of published namespaces will impact your Cognos 8 content. This is because changing the name of the namespace changes the IDs of the objects published in the metadata. Because the namespace is used as part of the object ID in Cognos 8, each namespace must have a unique name in the model. Each object in a namespace must also have a unique name. Part of the strategy of star schema groups is placing shortcuts into a separate namespace, which automatically creates a unique ID for each object in the namespace. This allows us to use the same name for shortcuts to conformed dimensions in different star schema groups.

The next time you try to run a query, report, or analysis against the updated model, you get an error. If you need to rename the namespace that you have published, use **Analyze Publish Impact** to determine which reports are impacted.

Folders are much simpler than namespaces. They are purely for organizational purposes and do not impact object IDs or your Cognos 8 content. You can create folders to organize objects by subject or functional area. This makes it easier for you to locate metadata, particularly in large projects.

The main drawback of folders is that they require unique names for all query subjects, dimensions, and shortcuts. Therefore, they are not ideal for containing shared objects such as conformed dimensions.

Dimensional Modeling Concepts

Regular and measure dimensions are used to enable an OLAP presentation of metadata, drilling up and down, and a variety of OLAP functions. You must use star schema groups (one fact with multiple dimensions) if you want to use Analysis Studio with a relational data source.

When building your model, it is recommended that model regular dimensions and model measure dimensions be created based on a relational model in which star schema concepts have been applied.

While you can convert data source query subjects to data source dimensions, data source dimensions have limited functionality in comparison to query subjects or model dimensions, and they are not recommended for general use.

Regular Dimensions

Regular dimensions represent descriptive data that provides context for data modeled in measure dimensions. A regular dimension is broken into groups of information called levels. In turn, the various levels can be organized into hierarchies. For example, a product dimension can contain the levels Product Line, Product Type, and Product organized in a single hierarchy called Product. Another example is a time dimension that has the levels Year, Quarter, Month, Week, and Day, organized into two hierarchies. The one hierarchy YQMD contains the levels Year, Quarter, Month, and Day, and the other hierarchy YWD contains the levels Year, Week, and Day.

The simplest definition of a level consists of a business key and a caption, each of these referring to one query item. An instance (or row) of a level is defined as a member of that level. It is identified by a member unique name, which contains the values of the business keys of the current and higher levels. For example, `[gosales].[Products].[ProductsOrg].[Product]->[All Products].[1].[1].[2]` identifies a member that is on the fourth level, `Product`, of the hierarchy `ProductsOrg` of the dimension `[Products]` that is in the namespace `[gosales]`. The caption for this product is TrailChef Canteen, which is the name shown in the metadata tree and on the report.

The level can be defined as unique if the business key of the level is sufficient to identify each set of data for a level. In the GO Sales sample model, the members of the Product level do not require the definition of Product type because there are no product numbers assigned to many different product types. A level that is not defined as unique is similar to a determinant that uses multiple-part keys because keys from higher levels of granularity are required (p. 326). If members within ancestor members are not unique but the level is defined as unique, data for the non-unique members is reported as a single member. For example, if City is defined as unique and identified by name, data for London, England and London, Canada will be combined.

A regular dimension may also have multiple hierarchies; however, you can use only one hierarchy at a time in a query. For example, you cannot use one hierarchy in the rows of a crosstab report and another hierarchy from the same dimension in the columns. If you need both hierarchies in the same report, you must create two dimensions, one for each hierarchy.

Measure Dimensions

Measure dimensions represent the quantitative data described by regular dimensions. Known by many terms in various OLAP products, a measure dimension is simply the object that contains the fact data. Measure dimensions differ from fact query subjects because they do not include the foreign keys used to join a fact query subject to a dimensional query subject. This is because the measure dimension is not meant to be joined as if it were a relational data object. For query generation purposes, a measure dimension derives its relationship to a regular dimension through the underlying query subjects. Similarly the relationship to other measure dimensions is through regular dimensions that are based on query subjects built to behave as conformed dimensions. To enable multiple-fact, multiple-grain querying, you must have query subjects and determinants created appropriately before you build regular dimensions and measure dimensions.

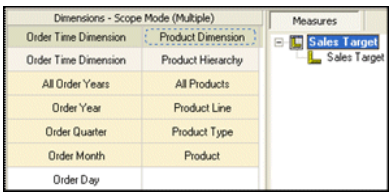
Scope Relationships

Scope relationships exist only between measure dimensions and regular dimensions to define the level at which the measures are available for reporting. They are not the same as joins and do not

impact the `Where` clause. There are no conditions or criteria set in a scope relationship to govern how a query is formed, it specifies only if a dimension can be queried with a specified fact. The absence of a scope relationship results in an error at runtime.

If you set the scope relationship for the measure dimension, the same settings apply to all measures in the measure dimension. If data is reported at a different level for the measures in the measure dimension, you can set scope on a measure. You can specify the lowest level that the data can be reported on.

In this example, the Sales Target measure dimension has only one measure that is in scope to the Order Month level on the Order Time Dimension and to the Product level of the Product Dimension. This means that if your users try to drill beyond the month level, they will see repeated data.



Building the Relational Model

Dimensional modeling of relational data sources is available in Cognos 8 Framework Manager, however it depends on the existence of a sound relational model. Cognos ReportNet provided some dimensional capabilities to enable multiple-fact querying and to prevent double-counting. Cognos 8 introduced features designed explicitly for dimensional representation of metadata and OLAP capability with relational data sources. The concepts applied to relational modeling in Cognos ReportNet are preserved in Cognos 8 with a few changes that are documented in the Framework Manager *User Guide*.

When you create a new model in Cognos 8, you will follow a common set of steps to define query generation even if you do not intend to use dimensional modeling capabilities. You must dimensionally model a relational data source when you want to use it in Analysis Studio, to enable drilling up and down in reports, or to access member functions in the studios.

When you build the relational model, we recommend that you do the following:

- Define the relational modeling foundation (p. 337).
- Define the dimensional representation of the model (p. 344), if it is required.
- Organize the model (p. 347).

Defining the Relational Modeling Foundation

A model is the set of related objects required for one or more related reporting applications. A sound relational model is the foundation for a dimensional model.

We recommend that you do the following when you define the relational modeling foundation:

- Import the metadata. For information about importing, see the Framework Manager *User Guide*.

- ❑ Verify the imported metadata (p. 338).
- ❑ Resolve ambiguous relationships (p. 338).
- ❑ Simplify the relational model using star schema concepts by analyzing cardinality for facts and dimensions and by deciding where to put relationships and determinants (p. 330).
- ❑ Add data security, if required. For information about data security, see the Framework Manager *User Guide*.

Then you can define the dimensional representation of the model (p. 344) if it is required, and organize the model for presentation (p. 347).

Verifying Imported Metadata

After importing metadata, you must check the imported metadata in these areas:

- relationships and cardinality
- determinants
- the **Usage** property for query items
- the **Regular Aggregate** property for query items

Relationships and cardinality are discussed here. For information on the **Usage** and **Regular Aggregate** properties, see the Framework Manager *User Guide*.

Analyzing the Cardinality for Facts and Dimensions

The cardinality of a relationship defines the number of rows of one table that is related to the rows of another table based on a particular set (or join) of keys. Cardinality is used by Cognos 8 to infer which query subjects behave as facts or dimensions. The result is that Cognos 8 can automatically resolve a common form of loop join that is caused by star schema data when you have multiple fact tables joined to a common set of dimension tables.

To ensure predictable queries, it is important to understand how cardinality is used and to correctly apply it in your model. It is recommended that you examine the underlying data source schema and address areas where cardinality incorrectly identifies facts or dimensions that could cause unpredictable query results. The **Model Advisor** feature in Framework Manager can be used to help you understand how Cognos 8 will interpret the cardinality you have set.

For more information, see "Cardinality" (p. 322).

Resolving Ambiguous Relationships

Ambiguous relationships occur when the data represented by a query subject or dimension can be viewed in more than one context or role, or can be joined in more than one way. The most common ambiguous relationships are:

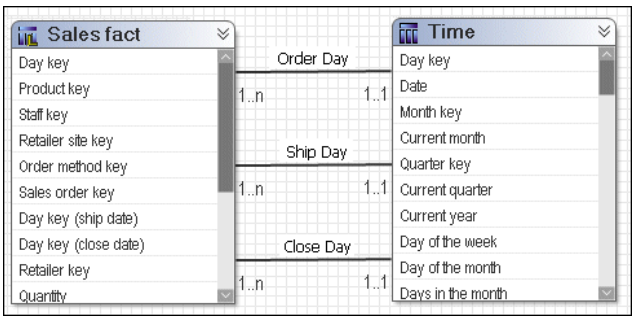
- role-playing dimensions (p. 339)
- loop joins (p. 341)
- reflexive and recursive relationships (p. 342)

You can use the **Model Advisor** to highlight relationships that may cause issues for query generation and resolve them in one of the ways described below. Note that there are other ways to resolve issues than the ones discussed here. The main goal is to enable clear query paths.

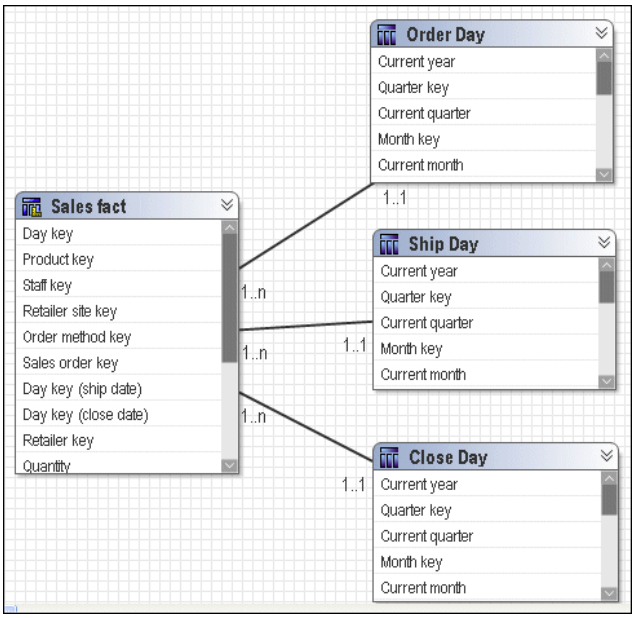
Role-Playing Dimensions

A table with multiple valid relationships between itself and another table is known as a role-playing dimension. This is most commonly seen in dimensions such as Time and Customer.

For example, the Sales fact has multiple relationships to the Time query subject on the keys Order Day, Ship Day, and Close Day.



Remove the relationships for the imported objects, fact query subjects, and role-playing dimensional query subjects. Create a model query subject for each role. Consider excluding unneeded query items to reduce the length of the metadata tree displayed to your users. Ensure that a single appropriate relationship exists between each model query subject and the fact query subject. **Note:** This will override the **Minimized SQL** setting but given a single table representation of the Time dimension, it is not considered to be problematic in this case.

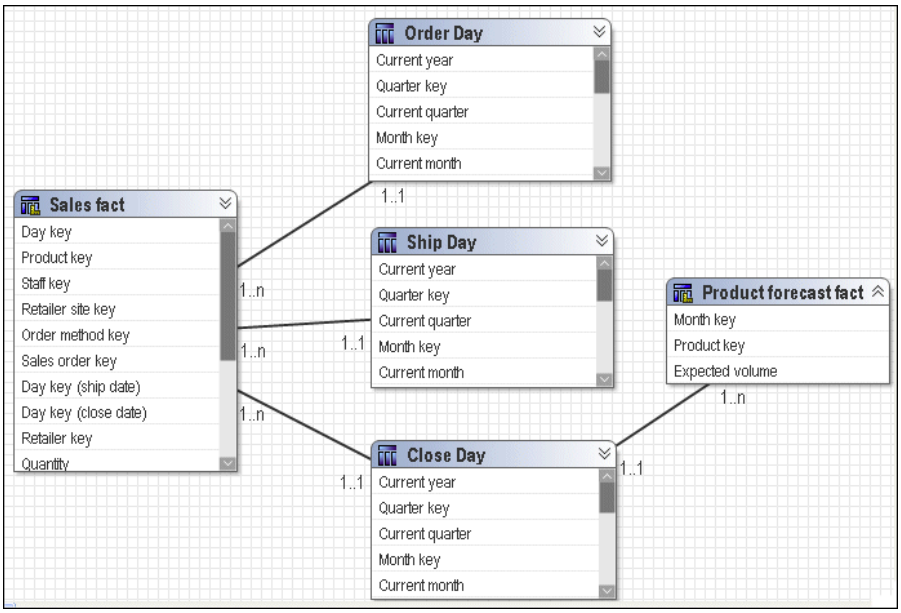


Decide how to use these roles with other facts that do not share the same concepts. For example, Product forecast fact has only one time key. You need to know your data and business to determine if all or any of the roles created for Time are applicable to Product forecast fact.

In this example, you can do one of the following:

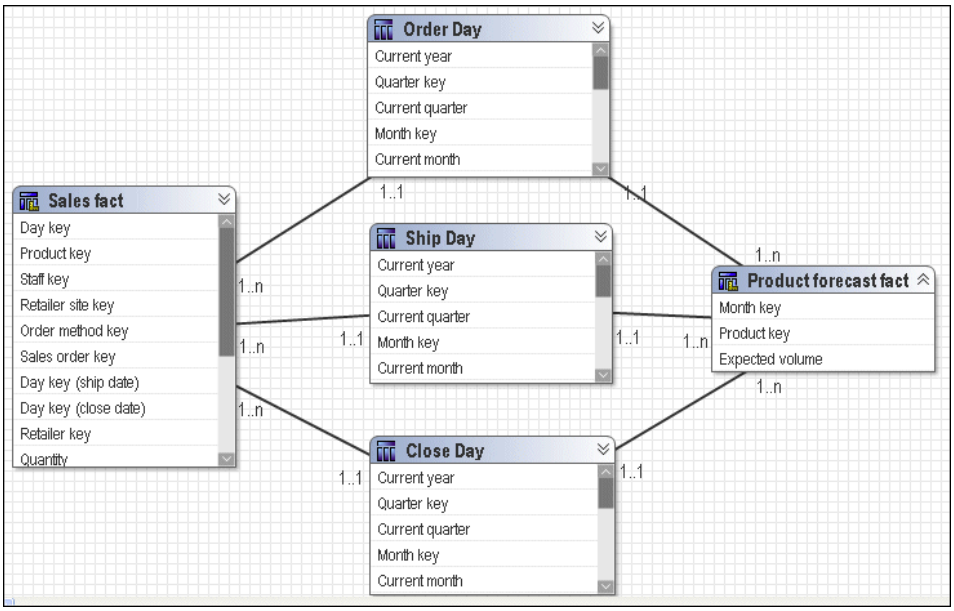
- Create an additional query subject to be the conformed time dimension and name it clearly as a conformed dimension.

  Pick the most common role that you will use. You can then ensure that this version is joined to all facts requiring it. In this example, Close Day has been chosen.



- You can treat Ship Day, Order Day, and Close Day as interchangeable time query subjects with Product forecast fact.

  In this case, you must create joins between each of the role-playing dimensions and Product forecast fact. You can use only one time dimension at a time when querying the Product forecast fact or your report may contain no data. For example, Month_key=Ship Month Key (200401) and Month key=Close Month Key (200312).



If modeling dimensionally, use each model query subject as the source for a regular dimension, and name the dimension and hierarchies appropriately. Ensure that there is a corresponding scope relationship specific to each role.

Loop Joins

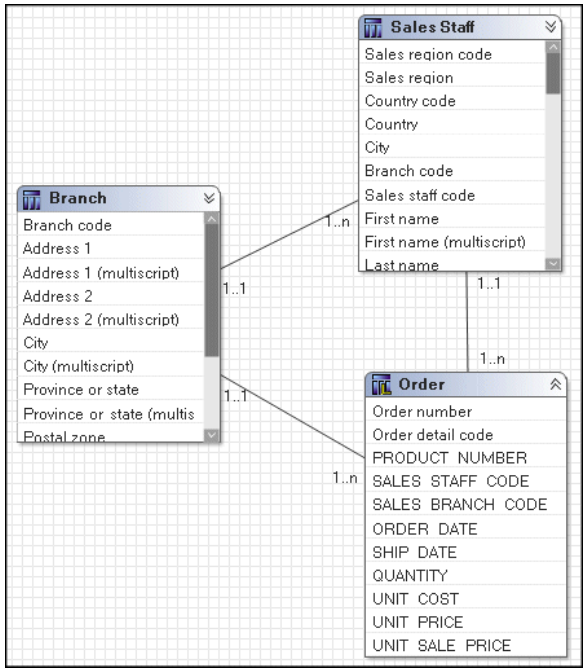
Loop joins in the model are typically a source of unpredictable behavior. This does not include star schema loop joins.

Note: When cardinality clearly identifies facts and dimensions, Cognos 8 can automatically resolve loop joins that are caused by star schema data when you have multiple fact tables joined to a common set of dimension tables.

In the case of loop joins, ambiguously defined query subjects are the primary sign of problems. When query subjects are ambiguously defined and are part of a loop join, the joins used in a given query are decided based on a number of factors, such as the location of relationships, the number of segments in join paths, and, if all else is equal, the alphabetically first join path. This creates confusion for your users and we recommend that you model to clearly identify the join paths.

Sales Staff and Branch provide a good example of a loop join with ambiguously defined query subjects.

In this example, it is possible to join Branch directly to Order or through Sales Staff to Order. The main problem is that when Branch and Order are together, you get a different result than when the join path is Branch to Sales Staff to Order. This is because employees can move between branches so employees who moved during the year are rolled up to their current branch even if many of the sales they made are attributable to their previous branch. Because of the way this is modeled, there is no guarantee which join path will be chosen and it is likely to vary depending on which items are selected in the query.

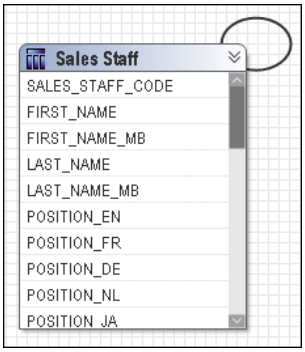


Reflexive and Recursive Relationships

Reflexive and recursive relationships imply two or more levels of granularity. Framework Manager imports reflexive relationships but does not use them when executing queries. Reflexive relationships, which are self-joins, are shown in the model for the purpose of representation only.

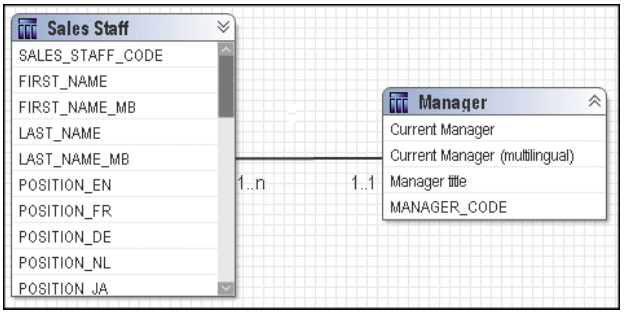
To create a functioning reflexive relationship, you can either create an alias shortcut, a copy of the data source query subject, or a model query subject. You then create a relationship between the original query subject and the new one. Using a model query subject tends to be the better option for flexibility because you can specify which query items are included in the query subject. Shortcuts are the better solution from a maintenance perspective. For more information, see "Model Objects vs. Shortcuts" (p. 333).

For example, the Sales Staff query subject has a recursive relationship between Staff Code and Manager Code.



Create a model query subject to represent Manager. Create a relationship with a `1..1` to `1..n` between Manager and Sales Staff. Then merge into a new model query subject.

For a simple two-level structure using a model query subject for Manager that is based on Sales Staff, the model looks like this:



For a recursive, balanced hierarchy, repeat this for each additional level in the hierarchy.

For a deep recursive or unbalanced hierarchy, we recommend that the hierarchy be flattened in the data source and that you model the flattened hierarchy in one regular dimension.

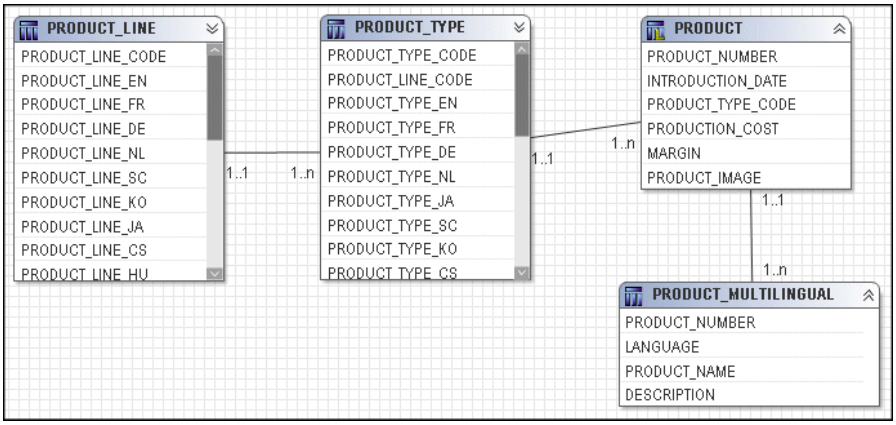
Simplifying the Relational Model

You can simplify the model by applying star schema concepts to the dimensional data and the fact data.

Modeling Query Subjects That Represent Descriptive Data

Cognos 8 dimensional modeling requires that you apply star schema principles to the logical layers of the model.

Normalized or snowflaked data sources often have several tables that describe a single business concept. For example, a normalized representation of Product may include four tables related by `1..n` relationships. Each Product Line has one or more Product Types. Each Product Type has one or more Products. Products have names and descriptions in multiple languages so they exist in the Product Multilingual lookup table.

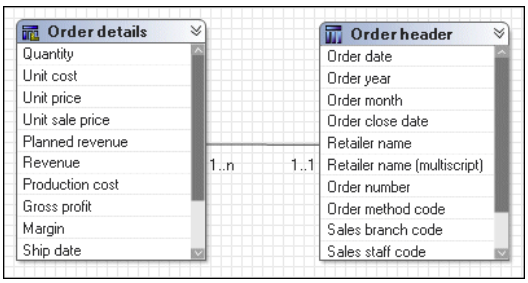


One way to simplify the model is to create one model query subject for each descriptive business concept. Your users may not know the relationship between the individual query subjects so it is helpful to group them together; in addition, having to expand each model object and select a query item requires more effort.

The next step for analysis is to create a regular dimension with a level for each query subject.

Modeling Fact Data

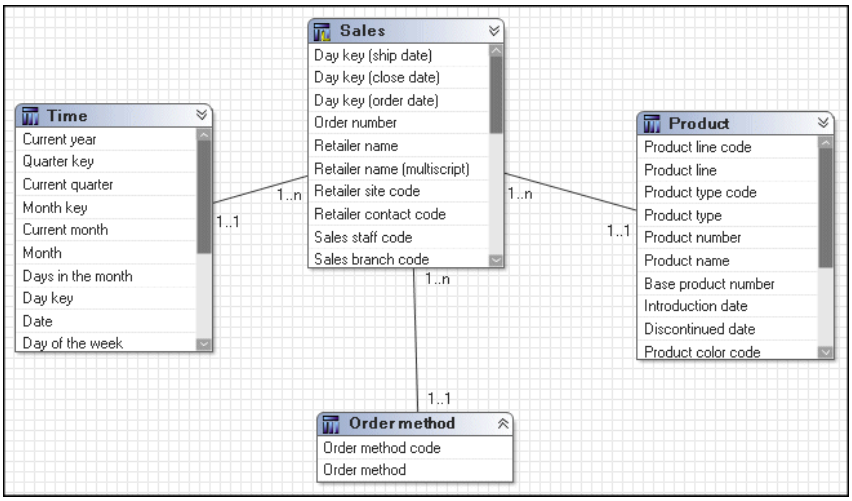
Data sources often have master-detail tables that contain facts. For example, when the Order header and Order details tables are used to insert and update data, the master-detail structure is beneficial. When these tables are used for reporting and analysis, you may choose to combine them into one logical business concept to simplify the model. Or you may choose to insert a dimension between them, such as Returned Items. Which solution you choose depends on your requirements.



To simplify the model in this example, apply star schema concepts to create one model query subject that combines the foreign keys of both Order header and Order details and includes all measures at the Order details level. This query subject should be joined to the same query subjects that Order header and Order details were joined to. You may choose to remove the original relationships from the two data source query subjects except for the relationship that defines the join between them.

For a discussion of the pros and cons of creating relationships to model query subjects, see the examples in "What Is Minimized SQL?" (p. 331).

In the example below, Order header and Order details have been combined into a new model query subject named Sales. This query subject has been joined to Product, Time, and Order method.



The next step for analysis is to create a measure dimension based on the model query subject.

Defining the Dimensional Representation of the Model

Dimensional modeling of relational data sources is a capability made available by Cognos 8 Framework Manager. You can model dimensions with hierarchies and levels and have facts with multiple measures. You can then relate the dimensions to the measures by setting scope in the model.

You must dimensionally model a relational data source when you want to use it in Analysis Studio, enable drilling up and down in reports, or access member functions in the studios.

We recommend that you use the relational model as the foundation layer (p. 337) and then do the following when you define the dimensional representation of the model:

- Create regular dimensions (p. 344).
- Model dimensions with multiple hierarchies (p. 345).
- Create measure dimensions (p. 346).
- Create scope relationships (p. 346).

Then you can organize the model for presentation (p. 347).

Creating Regular Dimensions

A regular dimension contains descriptive and business key information and organizes the information in a hierarchy, from the highest level of granularity to the lowest. It usually has multiple levels and each level requires a key and a caption. If you do not have a single key for your level, it is recommended that you create one in a calculation.

Model regular dimensions are based on data source or model query subjects that are already defined in the model. You must define a business key and a string type caption for each level. When you

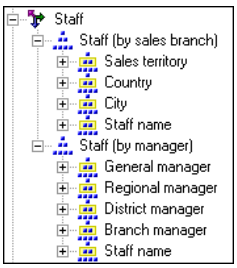
verify the model, the absence of business keys and caption information is detected. Instead of joining model regular dimensions to measure dimensions, create joins on the underlying query subjects and create a scope relationship between the regular dimension and the measure dimension.

Modeling Dimensions with Multiple Hierarchies

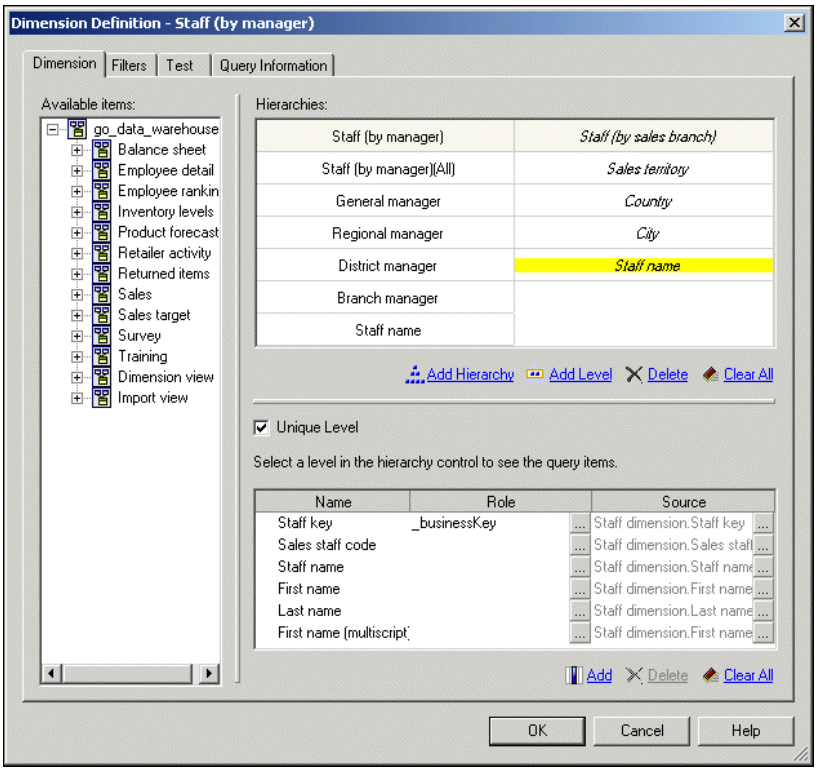
Multiple hierarchies occur when different structural views can be applied to the same data. Depending on the nature of the hierarchies and the required reports, you may need to evaluate the modeling technique applied to a particular case.

For example, sales staff can be viewed by manager or geography. In the Cognos 8 studios, these hierarchies are separate but interchangeable logical structures, which are bound to the same underlying query.

Here is sales staff as a single dimension with two hierarchies:



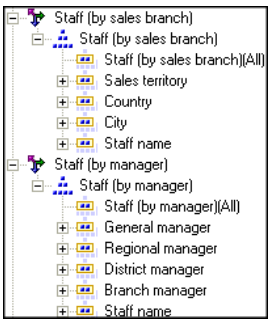
The hierarchies are defined in Framework Manager as follows.



You can specify multiple hierarchies on regular dimensions in Framework Manager. Multiple hierarchies for a regular dimension behave as views of the same query. However, you can use only one hierarchy at a time in a query. For example, you cannot use one hierarchy in the rows of a crosstab report and another hierarchy from the same dimension in the columns. If you need both

hierarchies in the same report, you must create two dimensions, one for each hierarchy. In cases where you have multiple hierarchies with significantly different levels or aggregation, you may choose to model so that a separate query subject with appropriate determinants exists as the foundation for that hierarchy. The only requirement is that any query subject used as the basis for a hierarchy must have a join defined to the query subject that provides the fact data.

Here are separate dimensions for each hierarchy.



Use this approach if dramatically different sets of columns are relevant for each hierarchy and it is more intuitive for your users to model the hierarchies as separate dimensions with separate and simpler queries.

Creating Measure Dimensions

A measure dimension is a collection of facts. You can create a measure dimension for one or more query subjects that have a valid relationship between them.

Model measure dimensions should be composed of only quantitative items. Because, by design, model measure dimensions do not contain keys on which to join, it is not possible to create joins to model measure dimensions. Instead of joining model measure dimensions to regular dimensions, create joins on the underlying query subjects. Then either manually create a scope relationship between them or detect scope if both dimensions are in the same namespace.

Create Scope Relationships

Scope relationships exist only between measure dimensions and regular dimensions to define the level at which the measures are available for reporting. They are not the same as joins and do not impact the `Where` clause. There are no conditions or criteria set in a scope relationship to govern how a query is formed, it specifies only if a dimension can be queried with a specified fact. The absence of a scope relationship results in an error at runtime.

If you set the scope relationship for the measure dimension, the same settings apply to all measures in the measure dimension. If data is reported at a different level for the measures in the measure dimension, you can set scope on a measure. You can specify the lowest level that the data can be reported on.

When you create a measure dimension, Framework Manager creates a scope relationship between the measure dimension and each existing regular dimension. Framework Manager looks for a join path between the measure dimension and the regular dimensions, starting with the lowest level of detail. If there are many join paths available, the scope relationship that Framework Manager creates may not be the one that you intended. In this case, you must edit the scope relationship.

Organizing the Model

After working in the relational modeling foundation (p. 337) and creating a dimensional representation (p. 344), we recommend that you do the following to organize the model:

- Keep the metadata from the data source in a separate namespace or folder.
- Create one or more optional namespaces or folders for resolving complexities that affect querying using query subjects.

  To use Analysis Studio, there must be a namespace or folder in the model that represents the metadata with dimensional objects.
- Create one or more namespaces or folders for the augmented business view of the metadata that contains shortcuts to dimensions or query subjects.

  Use business concepts to model the business view. One model can contain many business views, each suited to a different user group. You publish the business views.
- Create the star schema groups (p. 347).
- Apply object security, if required.
- Create packages and publish the metadata.

For information about the topics not covered here, see the Framework Manager *User Guide*.

Star Schema Groups

The concept of the conformed dimension is not isolated to dimensional modeling, it applies equally to query subjects.

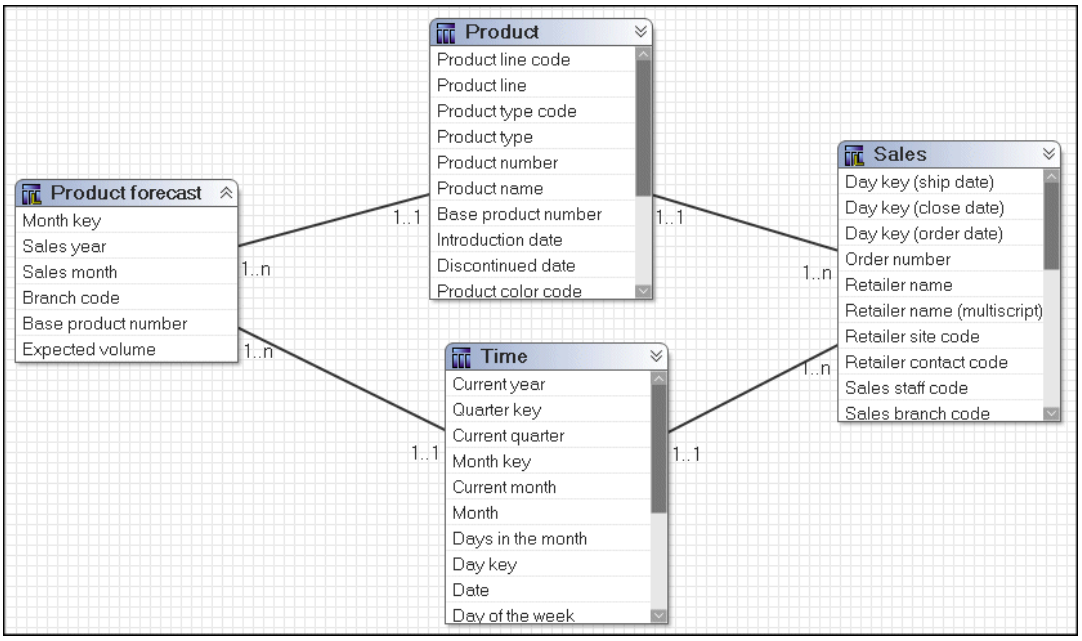
Use the **Star Schema Grouping** wizard to quickly create groups of shortcuts that will provide context for your users regarding which objects belong together. This makes the model more intuitive for your users. Star schema groups can also facilitate multiple-fact reporting by allowing the repetition of shared dimensions in different groups. This helps your users to see what different groups have in common and how they can do cross-functional, or multiple-fact, reporting. For more information, see "Multiple-fact, Multiple-grain Queries" (p. 327).

Star schema groups also provide context for queries with multiple join paths. By creating star schema groups in the business view of the model, you can clarify which join path to select when many are available. This is particularly useful for fact-less queries.

Multiple Conformed Star Schemas or Fact-less Queries

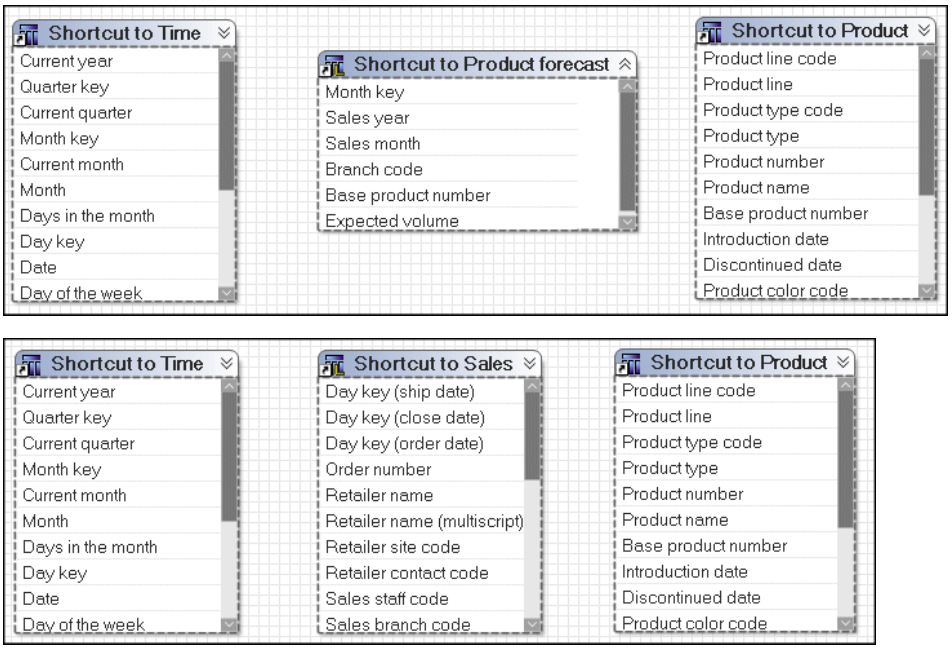
You will likely see dimensional query subjects that are joined to more than one fact query subject. Join ambiguity is an issue when you report using items from multiple dimensions or dimensional query subjects without including any items from the measure dimension or fact query subject. This is called a fact-less query.

For example, Product and Time are related to the Product forecast and Sales facts.



Using these relationships, how do you write a report that uses only items from Product and Time? The business question could be which products were forecasted for sale in 2005 or which products were actually sold in 2005. Although this query involves only Product and Time, these dimensions are related through multiple facts. There is no way to guess which business question is being asked. You must set the context for the fact-less query.

In this example, we recommend that you create two namespaces, one containing shortcuts to Product, Time, and Product forecast, and another containing Product, Time, and Sales.



When you do this for all star schemas, you resolve join ambiguity by placing shortcuts to the fact and all dimensions in a single namespace. The shortcuts for conformed dimensions in each namespace are identical and are references to the original object. **Note:** The exact same rule is applied to regular dimensions and measure dimensions.

With a namespace for each star schema, it is now clear to your users which items to use. To create a report on which products were actually sold in 2005, they use Product and Year from the Sales Namespace. The only relationship that is relevant in this context is the relationship between Product, Time, and Sales, and it is used to return the data.

Chapter 9: The SQL Generated by Cognos 8

The SQL generated by Cognos 8 is often misunderstood. This document explains the SQL that results in common situations.

Note: The SQL examples shown in this document were edited for length and are used to highlight specific examples.

This document is available in English, French, German, and Japanese. To access it in a different language, go to *installation location*\cognos\c8\webcontent\documentation and open the folder for the language you want. Then open ug_best.pdf.

Understanding Dimensional Queries

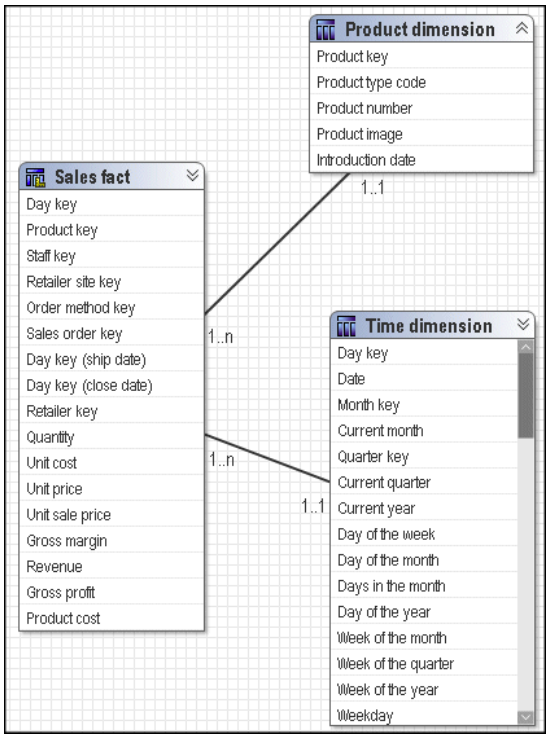
Dimensional queries are designed to enable multiple-fact querying. The basic goals of multiple-fact querying are:

- Preserve data when fact data does not perfectly align across common dimensions, such as when some fact data is missing for certain dimensional values but other facts are present.
- Prevent double-counting when fact data exists at different levels of granularity by ensuring that each fact is represented in a single query with appropriate grouping. Determinants may need to be created for the underlying query subjects in some cases.

Single Fact Query

A query on a star schema group results in a single fact query.

In this example, the Sales fact is the focus of any query that is written. The dimensional query subjects provide attributes and descriptions to make the data in Sales fact more meaningful. All relationships between dimensions and the fact are `1-n`.



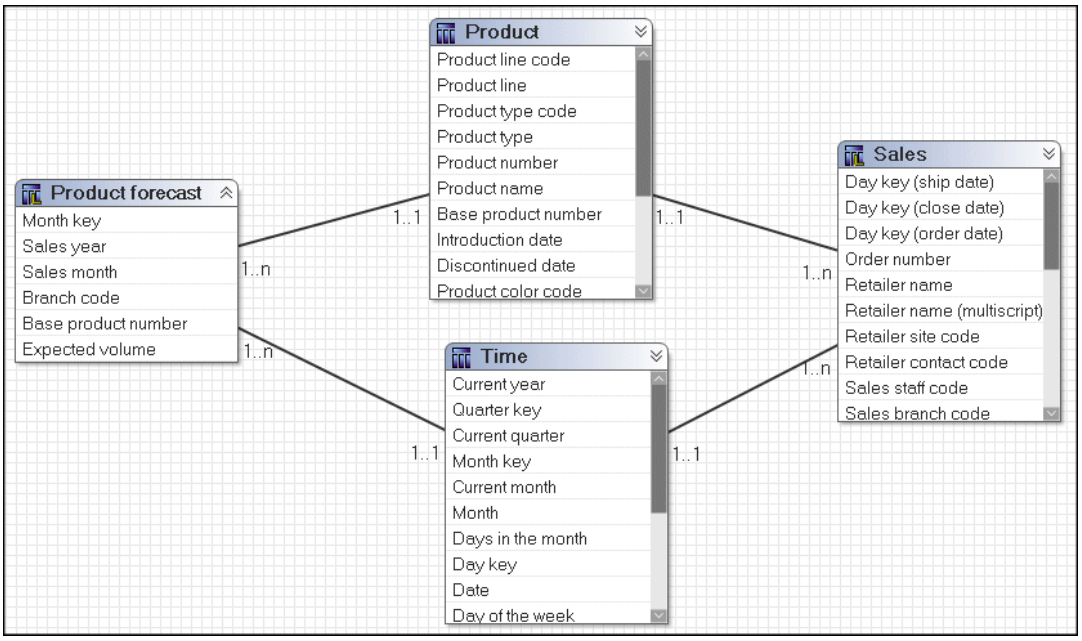
When you filter on the month and product, the result is as follows.

Month	Product	Quantity
January	TrailChef Water Bag	62,791
January	TrailChef Canteen	27,229
January	TrailChef Kitchen Kit	20,785
January	TrailChef Cup	29,670

Multiple-fact, Multiple-grain Query on Conformed Dimensions

A query on multiple facts and conformed dimensions respects the cardinality between each fact table and its dimensions and writes SQL to return all the rows from each fact table.

For example, Sales and Product forecast are both facts.



Note that this is a simplified representation and not an example of how this would appear in a model built using Cognos modeling recommendations.

The Result

Individual queries on Sales and Product forecast by Month and Product yield the following results. The data in Sales is actually stored at the day level.

Month	Product	Sales target
Opening balance		
January	TrailChef Water Bag	4,053,900
January	TrailChef Canteen	4,053,900
January	TrailChef Kitchen Kit	4,053,900
January	TrailChef Cup	4,053,900

A query on Sales and Product forecast respects the cardinality between each fact table and its dimensions and writes SQL to return all the rows from each fact table. The fact tables are matched on their common keys, month and product, and, where possible, are aggregated to the lowest common level of granularity. In this case, days are rolled up to months. Nulls are often returned for this type of query because a combination of dimensional elements in one fact table may not exist in the other.

Month	Product name	Quantity	Expected volume
April 2007	Aloe Relief	1,410	1,690
April 2007	Course Pro Umbrella	132	125
February 2007	Aloe Relief	270	245
February 2007	Course Pro Umbrella		1
February 2006	Aloe Relief	88	92

Note that in February 2007, Course Pro Umbrellas were in the forecast but there were no actual sales. The data in Sales and Product forecast exist at different levels of granularity. The data in Sales is at the day level, and Product forecast is at the month level.

The SQL

The SQL generated by Cognos 8, known as a stitched query, is often misunderstood. A stitched query uses multiple subqueries, one for each star, brought together by a full outer join on the common keys. The goal is to preserve all dimensional members occurring on either side of the query.

The following example was edited for length and is used to illustrate the main features of stitched queries.

```
select
 coalesce(D2.MONTH_NAME,D3.MONTH_NAME) as MONTH,
 coalesce(D2.PRODUCT_NAME,D3.PRODUCT_NAME) as PRODUCT,
 D2.EXPECTED_VOLUME as SALES_TARGET,
 D3.QUANTITY as QUANTITY
from (select TIME.MONTH_NAME as MONTH,
 PRODUCT_LOOKUP.PRODUCT_NAME as PRODUCT,
 XSUM(PRODUCT_FORECAST_FACT.EXPECTED_VOLUME for
 TIME.CURRENT_YEAR,TIME.QUARTER_KEY,TIME.MONTH_KEY,
 PRODUCT.PRODUCT_LINE_CODE, PRODUCT.PRODUCT_TYPE_CODE,
 PRODUCT.PRODUCT_KEY) as SALES_TARGET
from
 (select TIME.CURRENT_YEAR as CURRENT_YEAR,
 TIME.QUARTER_KEY as QUARTER_KEY,
 TIME.MONTH_KEY as MONTH_KEY,
 XMIN(TIME.MONTH_NAME for TIME.CURRENT_YEAR,
 TIME.QUARTER_KEY,TIME.MONTH_KEY) as MONTH
 from TIME_DIMENSION TIME
 group by TIME.MONTH_KEY) TIME
 join PRODUCT_FORECAST_FACT PRODUCT_FORECAST_FACT
 on (TIME.MONTH_KEY = PRODUCT_FORECAST_FACT.MONTH_KEY)
 join PRODUCT PRODUCT on (PRODUCT.PRODUCT_KEY =
 PRODUCT_FORECAST_FACT.PRODUCT_KEY)
where
 (PRODUCT.PRODUCT_NAME in ('Aloe Relief','Course Pro
Umbrella')) and
 (TIME.MONTH_NAME in ('April 2007','February 2007','February 2006'))
group by
 TIME.MONTH_NAME,
 PRODUCT_LOOKUP.PRODUCT_NAME
) D2
```

```
 full outer join
(select TIME.MONTH_NAME as MONTH_NAME,
 PRODUCT_LOOKUP.PRODUCT_NAME as PRODUCT_NAME,
 XSUM(SALES_FACT.QUANTITY for TIME.CURRENT_YEAR,
 TIME.QUARTER_KEY, TIME.MONTH_KEY,
 PRODUCT.PRODUCT_LINE_CODE, PRODUCT.PRODUCT_TYPE_CODE,
 PRODUCT.PRODUCT_KEY ) as QUANTITY
from
select TIME.DAY_KEY,TIME.MONTH_KEY,TIME.QUARTER_KEY,
 TIME.CURRENT_YEAR,TIME.MONTH_EN as MONTH_NAME
 from TIME_DIMENSION TIME) TIME
 join SALES_FACT SALES_FACT
 on (TIME.DAY_KEY = SALES_FACT.ORDER_DAY_KEY)
 join PRODUCT PRODUCT on (PRODUCT.PRODUCT_KEY = SALES_FACT.PRODUCT_KEY)
where
 PRODUCT.PRODUCT_NAME in ('Aloe Relief','Course Pro Umbrella'))
 and (TIME.MONTH_NAME in ('April 2004','February 2004','February
2006'))
 group by
 TIME.MONTH_NAME,
 PRODUCT.PRODUCT_NAME
) D3
 on ((D2.MONTH_NAME = D3.MONTH_NAME) and
 (D2.PRODUCT_NAME = D3.PRODUCT_NAME))
```

What Is the Coalesce Statement?

A `coalesce` statement is simply an efficient means of dealing with query items from conformed dimensions. It is used to accept the first non-null value returned from either subquery that is built for each detected fact table. This statement allows a full list of keys with no repetitions when doing a full outer join.

Why Is There a Full Outer Join?

A full outer join is necessary to ensure that all the data from each fact table is retrieved. An inner join gives results only if an item in inventory was sold. A right outer join gives all the sales where the items were in inventory. A left outer join gives all the items in inventory that had sales. A full outer join is the only way to learn what was in inventory and what was sold.

Modeling 1-n Relationships as 1-1 Relationships

If a `1-n` relationship exists in the data but is modeled as a `1-1` relationship, SQL traps cannot be avoided because the information provided by the metadata to Cognos 8 is insufficient.

The most common problems that arise if `1-n` relationships are modeled as `1-1` are the following:

- Double-counting for multiple-grain queries is not automatically prevented.

  Cognos 8 cannot detect facts and then generate a stitched query to compensate for double-counting, which can occur when dealing with hierarchical relationships and different levels of granularity across conformed dimensions.

- Multiple-fact queries are not automatically detected.

  Cognos 8 will not have sufficient information to detect a multiple-fact query. For multiple-fact queries, an inner join is performed and the loop join is eliminated by dropping the last evaluated

join. Dropping a join is likely to lead to incorrect or unpredictable results depending on the dimensions and facts included in the query.

If the cardinality were modified to use only `1-1` relationships between query subjects or dimensions, the result of a query on Product forecast and Sales with Time or Time and Product generates a single `Select` statement that drops one join to prevent a circular reference.

The example below shows that the results of this query are incorrect when compared with the results of individual queries against Sales or Product forecast.

The results of individual queries are as follows.

Month	Product	Quantity
January	TrailChef Water Bag	62,791
January	TrailChef Canteen	27,229
January	TrailChef Kitchen Kit	20,785
January	TrailChef Cup	29,670

Month	Product	Sales target
January	TrailChef Water Bag	4,053,900
January	TrailChef Canteen	4,053,900
January	TrailChef Kitchen Kit	4,053,900
January	TrailChef Cup	4,053,900

When you combine these queries into a single query, the results are as follows.

Month	Product	Quantity	Sales target
January	TrailChef Water Bag	62,791	4,053,900
January	TrailChef Canteen	27,229	4,053,900
January	TrailChef Kitchen Kit	20,785	4,053,900
January	TrailChef Cup	29,670	4,053,900

The SQL

If you look at the SQL, you can see that, because Cognos 8 detected that a circular join path exists in the model, it did not include one of the relationships that was not necessary to complete the join path. In this example, the relationship between Time and Product forecast was dropped.

A circular join path rarely results in a query that produces useful results.

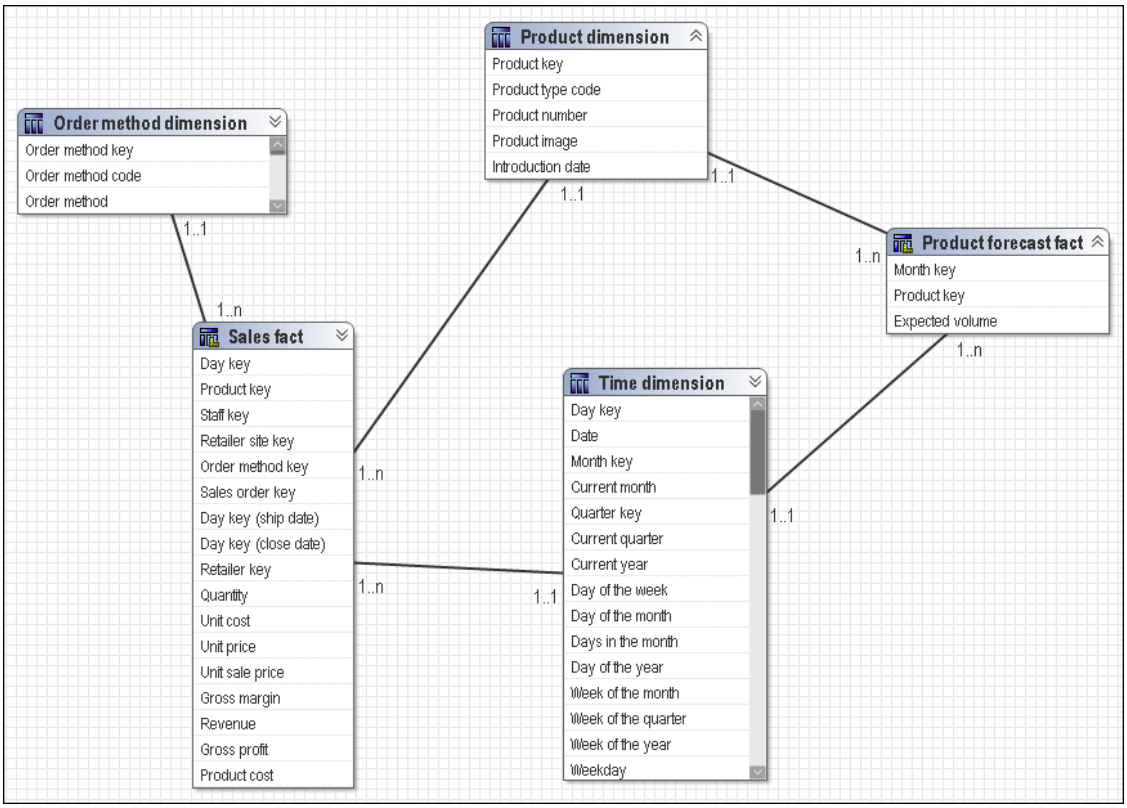
```
select
 TIME_.MONTH_NAME as MONTH,
 PRODUCT_LOOKUP.PRODUCT_NAME as PRODUCT,
 XSUM(SALES_FACT.QUANTITY for
 TIME_.CURRENT_YEAR, TIME_.QUARTER_KEY, TIME_.MONTH_KEY,
 PRODUCT.PRODUCT_LINE_CODE, PRODUCT.PRODUCT_TYPE_CODE,
 PRODUCT.PRODUCT_KEY ) as QUANTITY,
 XSUM(PRODUCT_FORECAST_FACT.EXPECTED_VOLUME for TIME_.CURRENT_YEAR,
 TIME_.QUARTER_KEY, TIME_.MONTH_KEY, PRODUCT.PRODUCT_LINE_CODE,
 PRODUCT.PRODUCT_TYPE_CODE, PRODUCT.PRODUCT_KEY ) as SALES_TARGET
from
 (select TIME.DAY_KEY,TIME.MONTH_KEY, TIME.QUARTER_KEY,
 TIME.CURRENT_YEAR,TIME.MONTH_EN as MONTH
 from TIME_DIMENSION TIME) TIME
 join
 SALES_FACT on (TIME_.DAY_KEY = SALES_FACT.ORDER_DAY_KEY)
 join
 PRODUCT_FORECAST_FACT on (TIME_.MONTH_KEY =
 PRODUCT_FORECAST_FACT.MONTH_KEY)
 join
 PRODUCT (PRODUCT.PRODUCT_KEY = PRODUCT_FORECAST_FACT.PRODUCT_KEY)
where
 (PRODUCT.PRODUCT_NAME in ('Aloe Relief','Course Pro Umbrella'))
and
 (TIME_.MONTH_NAME in ('April 2004','February 2004','February 2006'))
group by
 TIME_.MONTH_NAME, PRODUCT.PRODUCT_NAME
```

You can deal with these issues in Report Studio by creating an inner or outer join between queries or creating calculations that compensate for differences in granularity. This workaround requires the report author to have detailed knowledge of the data. There is no workaround available in Query Studio or Analysis Studio.

Please note that this approach can result in a query that joins across multiple fact tables in a single subquery, which can have a negative impact on performance.

Multiple-fact, Multiple-grain Query on Non-Conformed Dimensions

If a non-conformed dimension is added to the query, the nature of the result returned by the stitched query is changed. It is no longer possible to aggregate records to a lowest common level of granularity because one side of the query has dimensionality that is not common to the other side of the query. The result returned is really two correlated lists, which are related to each other only through the shared dimensional information.



The Result

The results of individual queries on the respective star schemas look like this.

Month	Product	Order method dimension	Quantity
February 2007	Aloe Relief	Telephone	55
February 2007	Aloe Relief	Email	41
March 2007	Aloe Relief	Web	1310
March 2007	Course Pro Umbrella	Fax	525
April 2007	Aloe Relief	Sales Visit	65
April 2007	Course Pro Umbrella	Sales Visit	46

Month	Product name	Quantity	Expected volume
April 2007	Aloe Relief	1,410	1,690
April 2007	Course Pro Umbrella	132	125

Month	Product name	Quantity	Expected volume
February 2007	Aloe Relief	270	245
February 2007	Course Pro Umbrella		1
February 2006	Aloe Relief	88	92

Querying the same items from both star schemas yields the following result.

Month	Product name	Order method dimension	Quantity	Expected volume
February 2007	Aloe Relief	Telephone	55	100,800
February 2007	Aloe Relief	Email	41	100,800
March 2007	Aloe Relief	Web	1310	113,200
March 2007	Course Pro Umbrella	Fax	525	1,554,800
April 2007	Aloe Relief	Sales Visit	65	113,200
April 2007	Course Pro Umbrella	Sales Visit	46	109,600

In this result, the lower level of granularity for records from Sales results in more records being returned for each month and product combination. There is now a `1-n` relationship between the rows returned from Product forecast and those returned from Sales.

When you compare this to the result returned in the example of the multiple-fact, multiple grain query on conformed dimensions, you can see that more records are returned and that Expected volume results are repeated across multiple Order Methods. Adding Order Method to the query effectively changes the relationship between Quantity data and Expected volume data to a `1-n` relationship. It is no longer possible to relate a single value from Expected volume to one value from Quantity.

Grouping on the Month key demonstrates that the result in this example is based on the same data set as the result in the multiple-fact, multiple-grain query but with a greater degree of granularity.

The SQL

The stitched SQL generated for this example is very similar to the SQL generated in the multiple-fact, multiple-grain query (p. 352). The main difference is the addition of Order Method. Order Method is not a conformed dimension and affects only the query against the Sales fact table.

```
select
 D2.QUANTITY as QUANTITY,
 D3.EXPECTED_VOLUME as SALES_TARGET,
 coalesce(D2.PRODUCT_NAME,D3.PRODUCT_NAME) as PRODUCT,
 coalesce(D2.MONTH_NAME,D3.MONTH_NAME) as MONTH,
 D2.ORDER_METHOD as ORDER_METHOD
from
 (select
 PRODUCT.PRODUCT_NAME as PRODUCT,
 TIME.MONTH_NAME as MONTH,
 ORDER_METHOD.ORDER_METHOD as ORDER_METHOD,
 XSUM(SALES_FACT.QUANTITY for TIME.CURRENT_YEAR,TIME.QUARTER_KEY,
 TIME.MONTH_KEY,PRODUCT.PRODUCT_LINE_CODE,PRODUCT.PRODUCT_TYPE_CODE,
 PRODUCT.PRODUCT_KEY,ORDER_METHOD_DIMENSION.ORDER_METHOD_KEY) as
QUANTITY
from
 PRODUCT_DIMENSION PRODUCT
 join
 SALES_FACT SALES_FACT
 on (PRODUCT.PRODUCT_KEY = SALES_FACT.PRODUCT_KEY)
 join
 ORDER_METHOD_DIMENSION ORDER_METHOD
 on (ORDER_METHOD.ORDER_METHOD_KEY = SALES_FACT.ORDER_METHOD_KEY)
 join TIME_DIMENSION TIME
 on ( TIME.DAY_KEY = SALES_FACT.ORDER_DAY_KEY)
where
 (PRODUCT.PRODUCT_NAME in ('Aloe Relief','Course Pro Umbrella'))
and
 ( TIME.MONTH_NAME in ('April 2004','February 2004','February 2006'))
 group by
 PRODUCT.PRODUCT_NAME,
 TIME.MONTH_NAME,
 ORDER_METHOD.ORDER_METHOD
 ) D2
 full outer join
(select
 PRODUCT.PRODUCT_NAME as PRODUCT,
 TIME.MONTH_NAME as MONTH,
 XSUM(PRODUCT_FORECAST_FACT.EXPECTED_VOLUME for TIME.CURRENT_YEAR,
 TIME.QUARTER_KEY,TIME.MONTH_KEY,PRODUCT.PRODUCT_LINE_CODE,
 PRODUCT.PRODUCT_TYPE_CODE,PRODUCT.PRODUCT_KEY) as EXPECTED_VOLUME
from
 PRODUCT_DIMENSION PRODUCT
 join
 PRODUCT_FORECAST_FACT PRODUCT_FORECAST_FACT
 on (PRODUCT.PRODUCT_KEY = PRODUCT_FORECAST_FACT.PRODUCT_KEY)
 join
(select
 TIME.CURRENT_YEAR as CURRENT_YEAR,
 TIME.QUARTER_KEY as QUARTER_KEY,
 TIME.MONTH_KEY as MONTH_KEY,
 XMIN(TIME.MONTH_NAME for TIME.CURRENT_YEAR, TIME.QUARTER_KEY,
 TIME.MONTH_KEY) as MONTH_NAME
from
 TIME_DIMENSION TIME
group by
 TIME.CURRENT_YEAR,
 TIME.QUARTER_KEY,
 TIME.MONTH_KEY
 ) TIME
 on (TIME.MONTH_KEY = PRODUCT_FORECAST_FACT.MONTH_KEY)
where
 (PRODUCT.PRODUCT_NAME in ('Aloe Relief','Course Pro Umbrella'))
and
```

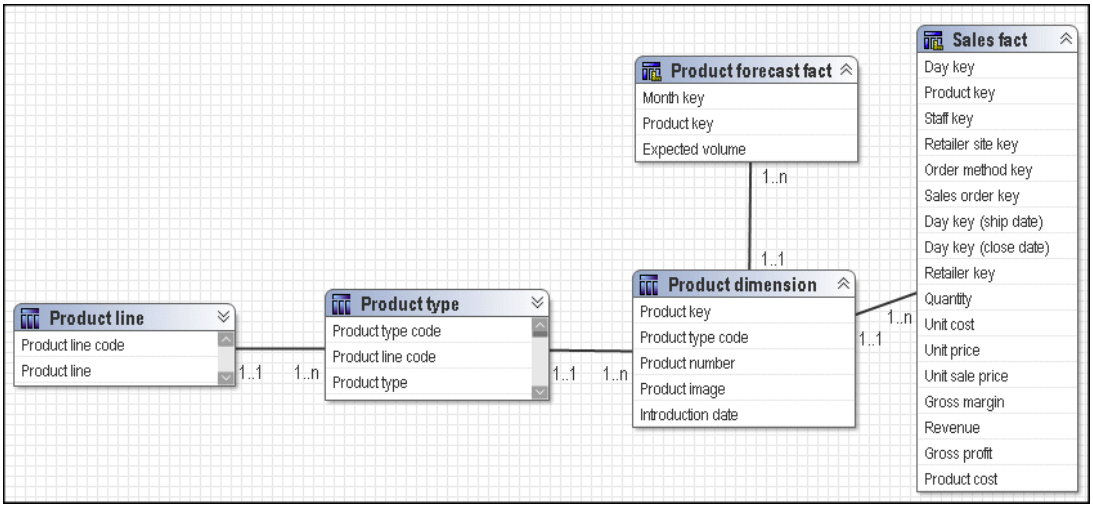
```
(TIME.MONTH_NAME in ('April 2004','February 2004','February 2006'))
group by
PRODUCT.PRODUCT_NAME,
TIME.MONTH_NAME
) D3
on ((D2.PRODUCT_NAME = D3.PRODUCT_NAME) and
(D2.MONTH_NAME = D3.MONTH_NAME))
```

Resolving Ambiguously Identified Dimensions and Facts

A query subject is considered to be ambiguously defined if it participates in both n and 1 relationships to other query subjects. An ambiguously defined query subject is not always harmful from a query generation perspective. We suggest that you evaluate query subjects using the following cases. The goal of this evaluation is to prevent unnecessary query splits and to ensure that any splits that do occur are intentional and correct.

Query Subjects That Represent a Level of Hierarchy

One frequent case of an ambiguously defined query subject that is not harmful is where the query subject represents an intermediate level of a descriptive hierarchy. One example is the Product hierarchy found in the Go Sales sample model.



In this example, both Product type and Product dimension could be identified as being ambiguously defined. However, this ambiguity is not detrimental to either the results generated or the performance of any query using one or more of these query subjects. You do not need to fix this query pattern because, using the rules for fact detection, only one fact is identified in any query that combines an item from the Product forecast or Sales query subjects. It remains a best practice to collapse hierarchies into a single regular dimension when modeling for analysis purposes.

Some queries that can be written using this example include the following:

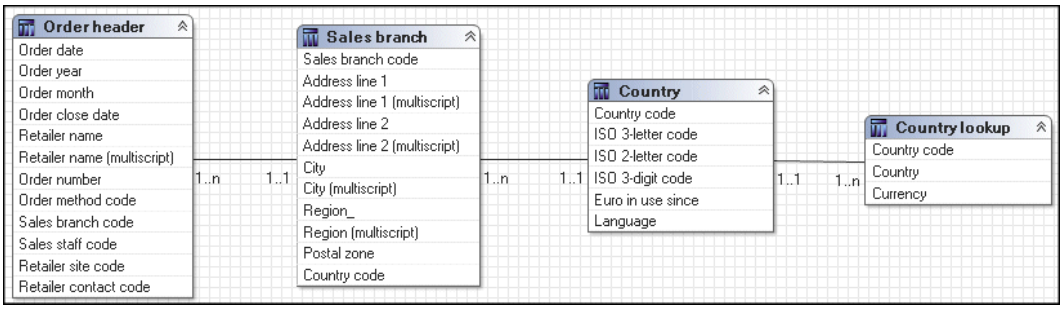
Items from these query subjects are used in a query:	Query subject that behaves as a fact in the query:
Product line and Product type	Product type
Product line, Product type, and Product dimension	Product dimension
Product line, Product type, Product dimension, and Sales fact	Sales fact
Product line and Sales fact	Sales fact
Product type and Product forecast fact	Product forecast fact

Resolving Queries That Should Not Have Been Split

If queries are split and should not be split, you must resolve these queries.

Query subjects on the n side of all relationships are identified as facts. We can see that in the following example, Order header and Country lookup are behaving as facts. In reality, the Country lookup query subject contains only descriptive information. From a business modeling perspective, Country lookup is an extension of Country.

Why is it a problem to leave the model like this?



Test this model by authoring a report on the number of orders per city, per country. Using this model returns an incorrect result. The numbers are correct for the cities but some cities are shown as being in the wrong country. This is an example of an incorrectly related result.

Country	City	Number of Orders
Australia	Arnoldstein	160
Austria	Calgary	205
Belgium	Heverlee	84

Country	City	Number of Orders
Canada	Melbourne	71
Switzerland	Zurich	80

To find the cause of this problem, you should start by looking at the SQL.

The SQL

In this example, we see a stitched query, which makes sense if we have multiple facts in the model. A stitched query is essentially a query that attempts to stitch multiple facts together. It uses the relationships that relate the facts to each other as well as the determinants for the conformed, or common, dimensions defined in the model. A stitched query can be identified by two queries with a full outer join. The wrapper query must include a `coalesce` statement on the conformed dimensions.

Note the following problems in the SQL:

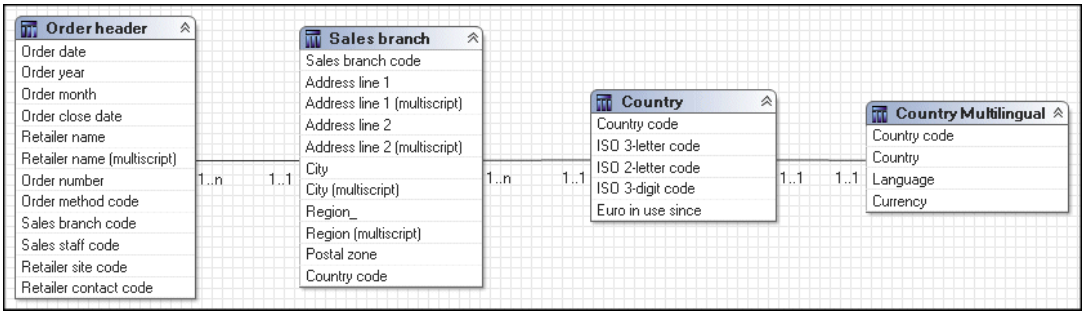
- The query has no `coalesce` statement.
- `RSUM` indicates an attempt to create a valid key.

```
select
 D3.COUNTRY as COUNTRY,
 D2.CITY as CITY,
 D2.number_of_orders as number_of_orders
from
(select
 SALES_BRANCH.CITY as CITY,
 XCOUNT(ORDER_HEADER.ORDER_NUMBER for SALES_BRANCH.CITY) as
 number_of_orders,
 RSUM(1 at SALES_BRANCH.CITY order by SALES_BRANCH.CITY
asc local)
 as sc
 from
 gosales.gosales.dbo.SALES_BRANCH SALES_BRANCH
 join
 gosales.gosales.dbo.ORDER_HEADER ORDER_HEADER
 on (SALES_BRANCH.SALES_BRANCH_CODE = ORDER_HEADER.SALES_BRANCH_CODE)
group by
 SALES_BRANCH.CITY
order by
 CITY asc
 ) D2
full outer join
(select
 COUNTRY_LOOKUP.COUNTRY as COUNTRY,
 RSUM(1 at COUNTRY_LOOKUP.COUNTRY order by
 COUNTRY_LOOKUP.COUNTRY asc local) as sc
from
 gosales.gosales.dbo.COUNTRY_LOOKUP COUNTRY_LOOKUP
group by
 COUNTRY_LOOKUP.COUNTRY
order by
 COUNTRY asc
 ) D3
 on (D2.sc = D3.sc)
```

By looking at the stitched columns in each query, we see that they are being calculated on unrelated criteria. This explains why there is no apparent relationship between the countries and cities in the report.

So why do we see a stitched query? To answer that question, we must look at the model.

In this example, the query items used in the report came from different query subjects. Country came from Country lookup, City came from Sales branch, and the Number of Orders came from a count on Order number in the Order header query subject.



The problem is that the query splits because the query engine sees this as a multiple-fact query. However, the split does not have a valid key on which to stitch because there is no item that both facts have in common.

There is more than one way to solve this problem but both require understanding the data.

Solution 1

You can add a filter to Country Multilingual that changes the cardinality of the relationship to `1-1`.

```
Select *
from [GOSL].COUNTRY_MULTILINGUAL
Where
COUNTRY_MULTILINGUAL."LANGUAGE"='EN'
```

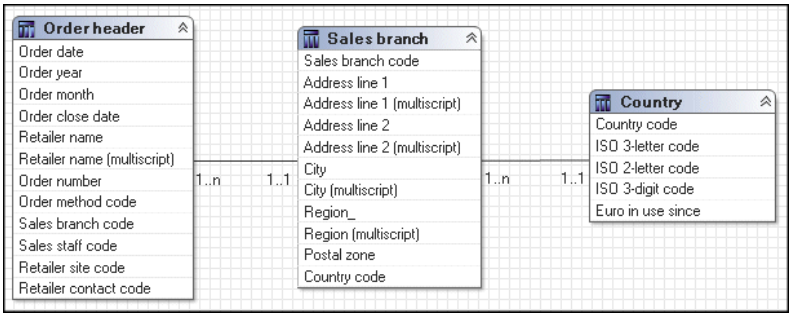
Or you can add a filter on the relationship and change the cardinality to `1-1`.

```
COUNTRY.COUNTRY_CODE = COUNTRY_MULTILINGUAL.COUNTRY_CODE
and COUNTRY_MULTILINGUAL.LANGUAGE = 'EN'
```

Either choice results in a model that has a single fact in this query.

Solution 2

Simplify the model by consolidating the related query subjects. This gives the greatest benefit by simplifying the model and reducing the opportunities for error in query generation.



With either solution, the result of the query is now correct.

Country	City	Number of Orders
Australia	Melbourne	160
Austria	Arnoldstein	205
Belgium	Heverlee	84
Canada	Calgary	71
Switzerland	Zurich	80

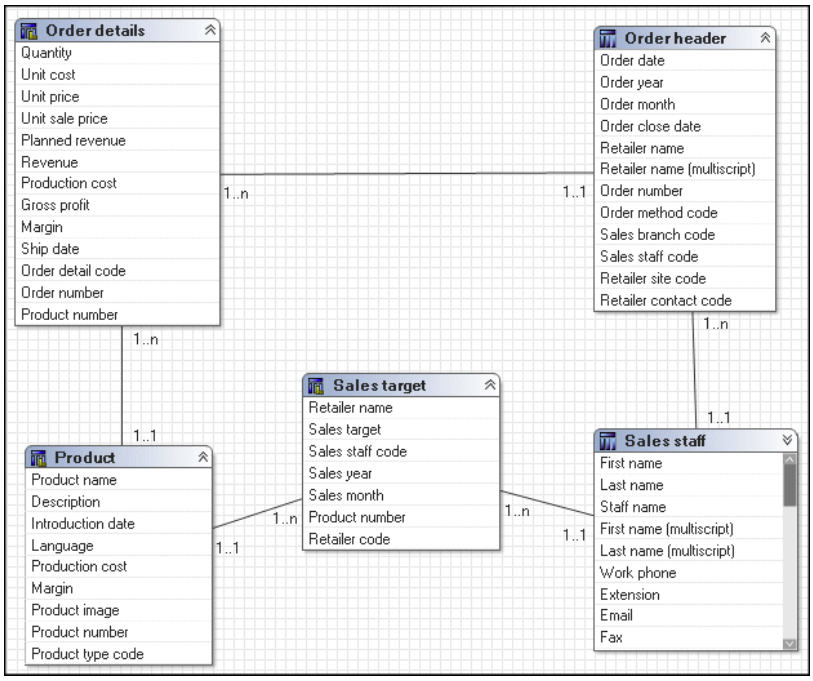
The SQL is no longer a stitched query.

```
select
 Country.c7 as COUNTRY,
 SALES_BRANCH.CITY as CITY,
 XCOUNT(ORDER_HEADER.ORDER_NUMBER for Country.c7,SALES_BRANCH.CITY)
 as number_of_orders
from
(select
 COUNTRY.COUNTRY_CODE as c1,
 COUNTRY_MULTILINGUAL.COUNTRY as c7
from
 gosales.gosales.dbo.COUNTRY COUNTRY
 join
 gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL
 on (COUNTRY.COUNTRY_CODE = COUNTRY_MULTILINGUAL.COUNTRY_CODE)
 where COUNTRY_MULTILINGUAL.LANGUAGE='EN'
 ) Country
join
 gosales.gosales.dbo.SALES_BRANCH SALES_BRANCH
 on (SALES_BRANCH.COUNTRY_CODE = Country.c1)
 join
 gosales.gosales.dbo.ORDER_HEADER ORDER_HEADER
 on (SALES_BRANCH.SALES_BRANCH_CODE = ORDER_HEADER.SALES_BRANCH_CODE)
group by
 Country.c7,
 SALES_BRANCH.CITY
```

Resolving Queries That Are Split in the Wrong Place

Sometimes using `1-n` cardinality does not resolve circular join paths. Incorrect results are still returned because of an improperly-formed stitched query.

For example, the model contains these objects.



If you report on Sales target and Quantity grouped by Product and Sales staff, the result set is incorrect. Sales target is too large.

Product	Last name	Sales target	Quantity
Husky Rope 50	Cartel	1,198,622	220
Aloe Relief	Cartel	69,821	422

To check the result, query each fact separately.

Product	Last name	Quantity
Husky Rope 50	Cartel	220
Aloe Relief	Cartel	422

Product	Last name	Sales target
Husky Rope 50	Cartel	180,600
Aloe Relief	Cartel	22,900

This confirms that the first report is giving a much larger result for Sales target than it should.

The SQL

Note the following problems when you look at the SQL:

- Only one of the dimension columns has a `coalesce` statement. This indicates that the query has been improperly split.
- Grouping is correct for the Quantity side of the query. This explains why Quantity is accurate.
- The quantity side of the stitched query is grouped only by Product, which explains why the Sales targets are too large.

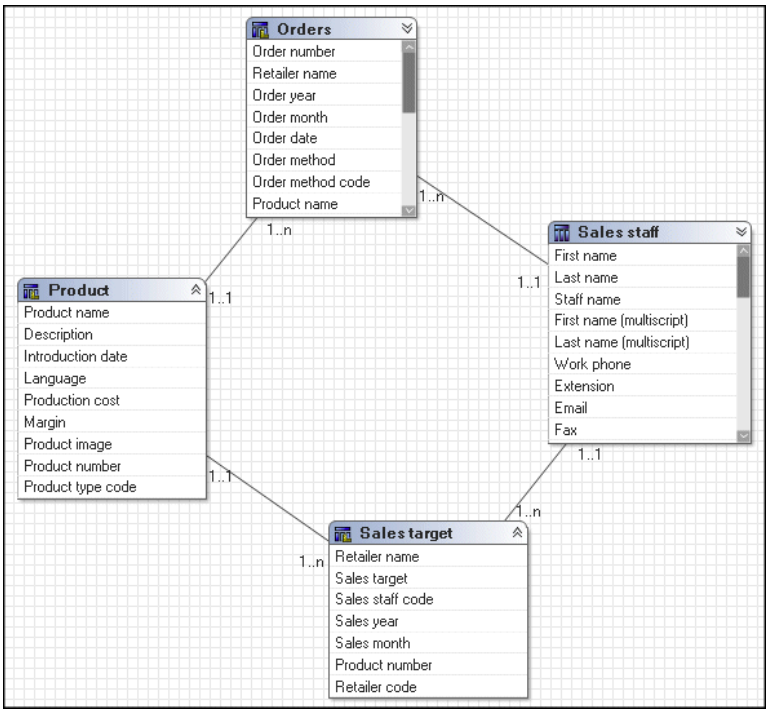
```
select
 coalesce(D2.PRODUCT_NAME,D3.PRODUCT_NAME) as PRODUCT,
 D2.LAST_NAME as LAST_NAME,
 D2.SALES_TARGET as SALES_TARGET,
 D3.Quantity as Quantity
from
(select
 Product.PRODUCT_NAME as PRODUCT,
 SALES_STAFF.LAST_NAME as LAST,
 XSUM(SALES_TARGET.SALES_TARGET for
 Product.PRODUCT_NAME,SALES_STAFF.LAST_NAME) as SALES_TARGET
from
 (select PRODUCT_MULTILINGUAL.PRODUCT_NAME,PRODUCT.PRODUCT_NUMBER
from
 gosales.gosales.dbo.PRODUCT PRODUCT,
 gosales.gosales.dbo.PRODUCT_MULTILINGUAL PRODUCT_MULTILINGUAL where
 PRODUCT.PRODUCT_NUMBER = PRODUCT_MULTILINGUAL.PRODUCT_NUMBER) Product
 join
 gosales.gosales.dbo.SALES_TARGET SALES_TARGET
 on (Product.PRODUCT_NUMBER = SALES_TARGET.PRODUCT_NUMBER)
 join
 gosales.gosales.dbo.SALES_STAFF SALES_STAFF
 on (SALES_STAFF.SALES_STAFF_CODE = SALES_TARGET.SALES_STAFF_CODE)
 group by
 Product.PRODUCT_NAME,
 SALES_STAFF.LAST_NAME
 ) D2
full outer join
 (select
 Product.PRODUCT_NAME as PRODUCT_NAME,
 XSUM(XSUM((ORDER_DETAILS.QUANTITY * ORDER_DETAILS.UNIT_SALE_PRICE)
 for Product.PRODUCT_NAME) at Product.PRODUCT_NAME
 for Product.PRODUCT_NAME)as Actual_sales
from
 (select PRODUCT_MULTILINGUAL.PRODUCT_NAME,PRODUCT.PRODUCT_NUMBER
from
 gosales.gosales.dbo.PRODUCT PRODUCT,
 gosales.gosales.dbo.PRODUCT_MULTILINGUAL
 PRODUCT_MULTILINGUAL where PRODUCT.PRODUCT_NUMBER =
 PRODUCT_MULTILINGUAL.PRODUCT_NUMBER) Product
 join
 gosales.gosales.dbo.ORDER_DETAILS ORDER_DETAILS
 on (Product.PRODUCT_NUMBER = ORDER_DETAILS.PRODUCT_NUMBER)
 group by
 Product.PRODUCT_NAME
 D3
 on (D2.PRODUCT_NAME = D3.PRODUCT_NAME)
```

These problems exist because nothing was selected from Order header. It was not included in the query so there was no way to link Sales staff to Order details. Linking Sales staff to Order details

was not necessary to connect all the tables because there was already a relationship between Sales staff and Sales target.

Although you can resolve this by including an item from Order header in the report, your users may not know to do that. We recommend that you fix it in the model. This is the recommended solution from both a dimensional and business modeling standpoint. In addition, a simpler model makes it harder to write a report that generates bad SQL.

Consolidate query subjects where logical groupings are possible. In this example, combine the Order details and Order header query subjects. The new group of query subjects can now be used for dimensional modeling.



When you use this new group of query subjects, the report looks like this.

Product	Last name	Sales target	Quantity
Husky Rope 50	Cartel	180,600	220
Aloe Relief	Cartel	22,900	422

When you look at the SQL, you see this:

```
select
 coalesce(D2.PRODUCT_NAME,D3.PRODUCT_NAME) as PRODUCT,
 coalesce(D2.LAST_NAME,D3.LAST_NAME) as LAST_NAME,
 D2.SALES_TARGET as SALES_TARGET,
 D3.Quantity as Quantity
from
 (select
 Product.PRODUCT_NAME as PRODUCT,
 SALES_STAFF.LAST_NAME as LAST_NAME,
 XSUM(SALES_TARGET.SALES_TARGET for
 Product.PRODUCT_NAME,SALES_STAFF.LAST_NAME) as SALES_TARGET
```

```
from
 (select PRODUCT_MULTILINGUAL.PRODUCT_NAME,PRODUCT.PRODUCT_NUMBER
from
 gosales.gosales.dbo.PRODUCT PRODUCT,
 gosales.gosales.dbo.PRODUCT_MULTILINGUAL
 PRODUCT_MULTILINGUAL where PRODUCT.PRODUCT_NUMBER =
 PRODUCT_MULTILINGUAL.PRODUCT_NUMBER) Product
 join
 gosales.gosales.dbo.SALES_TARGET SALES_TARGET
 on (SALES_TARGET.PRODUCT_NUMBER = Product.PRODUCT_NUMBER)
 join
 gosales.gosales.dbo.SALES_STAFF SALES_STAFF
 on (SALES_STAFF.SALES_STAFF_CODE = SALES_TARGET.SALES_STAFF_CODE)
group by
 Product.PRODUCT_NAME,
 SALES_STAFF.LAST_NAME
 ) D2
 full outer join
 (select
 Product.PRODUCT_NAME as PRODUCT,
 SALES_STAFF.LAST_NAME as LAST_NAME,
 XSUM((ORDER_DETAILS_ORDER_HEADER.c5 * ORDER_DETAILS_ORDER_HEADER.c8)
 for Product.PRODUCT_NAME,SALES_STAFF.LAST_NAME ) as Actual_sales
from
 (select PRODUCT_MULTILINGUAL.PRODUCT_NAME,PRODUCT.PRODUCT_NUMBER
from
 gosales.gosales.dbo.PRODUCT PRODUCT,
 gosales.gosales.dbo.PRODUCT_MULTILINGUAL
 PRODUCT_MULTILINGUAL where PRODUCT.PRODUCT_NUMBER =
 PRODUCT_MULTILINGUAL.PRODUCT_NUMBER) Product
 join
 (select
 ORDER_DETAILS.PRODUCT_NUMBER as c4,
 ORDER_DETAILS.QUANTITY as c5,
 ORDER_DETAILS.UNIT_SALE_PRICE as c8,
 ORDER_HEADER.SALES_STAFF_CODE as c14
from
 gosales.gosales.dbo.ORDER_DETAILS ORDER_DETAILS
 join
 gosales.gosales.dbo.ORDER_HEADER ORDER_HEADER
 on (ORDER_HEADER.ORDER_NUMBER = ORDER_DETAILS.ORDER_NUMBER)
 ) ORDER_DETAILS_ORDER_HEADER
 on (ORDER_DETAILS_ORDER_HEADER.c4 = Product.PRODUCT_NUMBER)
 join
 gosales.gosales.dbo.SALES_STAFF SALES_STAFF
 on (SALES_STAFF.SALES_STAFF_CODE = ORDER_DETAILS_ORDER_HEADER.c14)
group by
 Product.PRODUCT_NAME,
 SALES_STAFF.LAST_NAME
 ) D3
 on ((D2.PRODUCT_NAME = D3.PRODUCT_NAME) and (D2.LAST_NAME = D3.LAST_NAME)))
```

Chapter 10: Upgrading Models

When you install the latest version of Cognos 8, most of the upgrade of existing models is done automatically. There are some differences in what you must do depending on whether you are upgrading a Cognos ReportNet model (p. 371) or a Cognos 8.1 or 8.2 model (p. 378).

The packages you published for your users are automatically upgraded in the Content Manager database. These packages continue to function as they did previously and all reports continue to run. Upgrading the model in Framework Manager is required only if you want to modify the metadata that you published.

If a model cannot upgrade successfully, reports that use the model will not run. You must then open the model in Framework Manager, upgrade it manually, and republish it.

Cognos ReportNet Model Upgrade

Upgrading a model prepares the model to take advantage of the new features in Cognos 8.

To upgrade your model, follow these steps:

- Verify the model in your existing environment (p. 371).
- Open and upgrade the model (p. 372).
- Verify and repair the model (p. 374).

Verifying the Model in Cognos ReportNet

The upgrade process does not resolve any problems that existed in your model prior to upgrading. Before upgrading the model, we recommend that you verify your model and fix any reported problems in your existing environment.

When upgrading from Cognos ReportNet to Cognos 8.3, you must ensure that dimension information is set correctly in the Cognos ReportNet model (p. 371) and then verify the model and fix errors.

How Dimension Information Changed

In Cognos ReportNet, dimension information combined the concept of uniqueness with the concept of dimensional hierarchies. In Cognos 8, the concept of dimension information is divided between determinants and hierarchies for regular dimensions. Determinants control uniqueness and granularity. This control is required for relationally-based query subjects, particularly multiple-fact, multiple-grain queries (p. 327). Hierarchies for regular dimensions address dimensional concepts of hierarchies, levels, keys, and attributes for all data sources.

When you upgrade the model, you can use the dimension information to create query subjects with determinants or regular dimensions. Thus you must ensure that dimension information was specified

correctly in your ReportNet model. When reviewing dimension information, you must understand how the dimension information is applied to the query subject and how you will use the query subject in the Cognos 8 model.

Before upgrading the model to Cognos 8, we recommend that you review your ReportNet model using the *Modeling in Framework Manager for Predictable Queries and Results* paper, which is available from the Cognos support Web site (http://support.cognos.com).

Opening and Upgrading the Model

When you open an older model in Framework Manager, you are notified that the model must be updated.

When upgrading from Cognos ReportNet to Cognos 8.3, you should

- understand the impact of governors that are set during upgrade (p. 372)
- review the data types that have changed (p. 373)
- understand the impact on query subjects based on SAP BW metadata (p. 373)

If you are upgrading a model that is segmented or linked, see "Upgrading Segmented and Linked Projects" (p. 380).

Governors for a Cognos ReportNet Upgrade

Several governors are set automatically during the upgrade process:

- The **Allow enhanced portability at run time** governor is selected.

  This governor is used when upgrading a Cognos ReportNet 1.x model. It prevents rigid enforcement of data types so that a Cognos 8 model can function as a ReportNet 1.x model until you update the data types in the metadata. After you have verified that the model has been upgraded successfully, clear this governor.

  The status property of all data source query subjects is set to **Needs Re-evaluation**. The query engine ignores the data types in the model that are flagged as **Needs Re-evaluation** and retrieves the data type information from the data source. This may have a slightly negative impact on performance when running reports.

- The **Allow dynamic generation of dimension information** governor is selected.

  This governor ensures consistent behavior with Cognos ReportNet by deriving a form of dimension information from the relationships, key information, and index information in the data source.

  When this governor is selected and a model contains query subjects but no dimensions, Cognos 8 generates queries that avoid double-counting. This also provides backward compatibility with ReportNet. When you verify the model, you see a warning when this governor is selected.

- The **Shortcut Processing** governor is set to **Automatic**.

  This governor controls how shortcuts are processed by Cognos 8. When the governor is set to **Automatic**, the model works the same as in ReportNet, that is, a shortcut that exists in the same folder as its target behaves as an alias, or independent instance. However, a shortcut

existing elsewhere in the model behaves as a reference to the original. When you create a new model, the **Shortcut Processing** governor is always set to **Explicit**.

- The **Suppress Null values for SAP BW data sources** governor is selected for SAP BW metadata only.

  This controls the creation of outer joins in models based on SAP BW metadata.

  Some queries can be very large because null values are not filtered out. Null suppression removes a row or column for which all of the values in the row or column are null (empty). Null suppression is performed by SAP BW. This reduces the amount of data transferred to the Cognos 8 studios and improves performance.

Data Types

Cognos 8 added support for additional data types. This may change how data types are mapped between Cognos and the source data on metadata import.

Mapping of these data types varies by data source vendor.

The primary areas of change are as follows:

Data Type in Cognos ReportNet	Data Type in Cognos 8
char	nChar
decimal	numeric
varChar	nVarChar
varChar	timestampTZ
varChar	IntervalTZ

Data types are stored in the model so that Cognos 8 can avoid going to the data source to determine the data types for query items, resulting in improved performance.

Some query items may appear broken after the upgrade, particularly calculations. For example, the data type of the underlying items for a calculation changed so it is not possible to automatically assign a data type to the calculation. Another example is you saved a calculation that uses a prompt without testing it. For these cases, test the calculation or query item that is affected to correct the data type.

Query Subjects Based on SAP BW Metadata

A query subject that is based on SAP BW metadata is not converted to a dimension. Instead, the upgrade process creates a hidden dimension and a new model query subject that have the same name and query items as the original query subject. The model query subject points to the newly created hidden dimension, which is used to maintain compatibility with existing reports.

Prior to upgrading, conformed dimensions are represented by query subject shortcuts. During upgrade, the conformed dimensions are not altered. The query subject shortcuts are left intact, but the targeted SAP BW query subjects are replaced by model query subjects corresponding to the hidden dimension.

Model query subjects that do not contain levels and hierarchies are converted to measure dimensions.

After upgrading, review the model to ensure that the newly created regular and measure dimensions meet your analysis and reporting requirements. You should only conform dimensions as required to fulfill your reporting and analysis needs.

If you are developing a new application based on an existing model and you are not required to support existing content, you can delete conformed dimensions and re-import the objects from your SAP BW data source. This creates regular dimensions without the underlying query subjects.

Verifying and Repairing the Upgraded Cognos ReportNet Model

After updating the metadata, Framework Manager prompts you to verify the model. You will see at least one warning for each object that contains dimension information or one of the data types listed previously (p. 373). We recommend that you check one namespace at a time in large models. You can also check an individual object.

Verifying the model separately from the upgrade process offers a number of advantages. You can open an existing model and upgrade and publish without any changes in query functionality. You can take the time to reassess model requirements. You can continue to design the production model and publish it without dimensional information. For models created with Cognos ReportNet, you can move to dimensions and determinants gradually.

When upgrading from Cognos ReportNet to Cognos 8.3, you need

- to convert query subjects with dimension information to query subjects with determinants or to dimensions (p. 374)
- to select and repair objects (p. 377)

If you make changes to the model before verifying it, you should not change the determinants. Doing so may result in losing the dimension information, which will prevent a successful model verification later.

Converting Dimension Information to Determinants or Dimensions

In Cognos ReportNet, dimension information combined the concept of uniqueness with the concept of dimensional hierarchies. In Cognos 8, the concept of dimension information is divided between determinants and hierarchies for regular dimensions. Determinants control uniqueness and granularity. This control is required for relationally-based query subjects, particularly multiple-fact, multiple-grain queries. Hierarchies for regular dimensions address dimensional concepts of hierarchies, levels, keys, and attributes for all data sources.

Query subjects with dimension information are not converted to determinants or dimensions during the upgrade. The metadata previously specified by dimension information is preserved in the model and continues to exist for query subjects until those query subjects are repaired. You cannot change this dimension information in Cognos 8. You can upgrade the dimension information to determinants

or dimensions. Until you upgrade the query subject, Cognos 8 uses the dimension information previously specified in Cognos ReportNet.

If you decide to make changes to the model without verifying it after the upgrade, do not make changes to the determinants because you may lose the dimensional information.

Determinants

Use determinants if you want to preserve existing reports while extending your application.

Determinants reflect granularity by representing subsets or groups of data in a query subject and are used to ensure correct aggregation of this repeated data. Determinants are most closely related to the concept of keys and indexes in the data source and are imported based on unique key and index information in the data source.

Unlike dimension information, model query subjects inherit determinants that are specified for the underlying query subjects. We recommend that determinants be specified as early as possible in the model, typically on the data source query subject. In cases where different determinants are specified for different granularity control, you should either create multiple instances of the data source query subject or remove determinants from the data source query subject and create new determinants on each model query subject requiring different granularity control.

Dimensions

Use dimensions if you want to use your model in Analysis Studio, enable drilling up and down in reports, or access member functions in the studios. Only regular dimensions are created. During upgrade, sufficient metadata does not exist to create measure dimensions so you must create them manually. You can then relate the dimensions to the measures by setting scope in the model.

It is important to understand that if you upgrade query subjects to dimensions, the names of the query items in your model are changed. For example, a query item previously qualified as *namespace.query subject.query item* now has the name *namespace.dimension.hierarchy.level.query item*.

Because this name change is likely to invalidate existing reports, we recommend upgrading query subjects to dimensions only when you want to leverage an existing model to build a new application that requires dimensional capability.

How Dimension Information Is Mapped

Dimension information is mapped to determinants and dimensions as follows.

Dimension information	Determinants	Dimensions
Hierarchies	For relational data sources, the first hierarchy is used to create the determinant.	Hierarchies For SAP BW data sources, alternate hierarchies are upgraded to alternate hierarchies of a dimension.

Dimension information	Determinants	Dimensions
Levels	Determinants, either uniquely identified or group by	Levels The first level of the hierarchy is automatically defined as the All level. It contains a single root member, which represents the top level of the hierarchy. You cannot delete or move the All level. You can change its name, description, and screen tip.
Keys	If Unique Key is not selected, key segments from higher levels are included in the key.	_businessKey role
	If Unique Key is selected, only the key segment, or segments, for the level are included in the key.	Unique Level
Alphabetically first text attribute		_memberCaption role
Attributes	Attributes Unassociated attributes are assigned to the last determinant, which generally corresponds to the lowest level.	Can be manually assigned to be _memberDescription role, custom role, or no role

Things to Review After Converting

After converting query subjects with dimension information to query subjects with determinants, you must review the following:

- Uniquely Identified

  When you define a unique determinant, you are indicating that the key contains enough information to identify a group within the data.

- Group By

  When you define a non-unique determinant, you should specify group by. This indicates to Cognos 8 that, when the keys or attributes associated with that determinant are repeated in the data, it should apply aggregate functions and grouping to avoid double-counting.

- Multiple, or alternate, hierarchies that existed in Cognos ReportNet

If two hierarchies existed on a query subject in Cognos ReportNet, only the first hierarchy is upgraded to a determinant for relational data sources. You must create a second query subject and manually specify the determinants for the other hierarchy.

For SAP BW data sources, alternate hierarchies are upgraded to determinants.

After converting query subjects with dimension information to dimensions, you must review the following:

- _businessKey role

  This role represents the key for the level and can be assigned to only one attribute in a level.

- Unique Level

  A unique level indicates that the keys of the levels above are not necessary to identify the members in this level.

- _memberCaption role

  To leverage member functions in the Cognos 8 studios, you must assign a _memberCaption role to each level in a dimension. If there are no attributes for the level, the absence of a caption is highlighted when you verify the model.

  All captions must have the string data type. If there is no attribute of this type available, create a calculation that is a string data type and assign the _memberCaption role to the new item. This is primarily an issue for Analysis Studio.

- Attributes

  In general, all other attributes should be included in the dimension and associated to the correct level. By default, they are included with no role. You have the option to create custom roles or assign attributes to existing roles.

- Multiple Hierarchies

  A regular dimension may have multiple hierarchies; however, you can use only one hierarchy at a time in a query. For example, you cannot use one hierarchy in the rows of a crosstab report and another hierarchy from the same dimension in the columns. If you need both hierarchies in the same report, you must create two dimensions, one for each hierarchy.

Selecting and Repairing Objects in the Upgraded Cognos ReportNet Model

The repair process first evaluates all selected items. This evaluation automatically resolves issues with new data types and prompts you to repair dimension information in the model.

You can select one or more check boxes of items and repair them.

The following warnings may appear when you verify and repair an upgraded model:

Warning	Description
Needs reevaluation	This message is most often related to data type changes. The majority of items with this warning can be selected for repair. The repair option provides options for evaluating and upgrading specific elements of metadata.
Join expression conflicts with the determinant information defined in the query subject	Sometimes the index and key information specified for a query subject implies a level of granularity that does not match the relationships specified on a query subject.
None of the query items in this level have a caption role specified	When defining levels, you must ensure that a business key and caption roles are specified. These roles are needed for member functions in the Cognos 8 studios and to assist in the member-oriented tree in Analysis Studio. All captions must have the string data type. If there is no attribute of this type available, create a calculation that is a string data type and assign the member caption role to the new item. This is primarily an issue for Analysis Studio.
One or more determinants that describe the keys and attributes of the query subject should be specified	Determinants are based on key information in the data source. It is possible that no determinants exist for a query subject upgraded from Cognos ReportNet, especially for a model query subject. We recommend that you use determinants to specify the granularity of the data in the query subject and the functional dependencies between query items. However, it is not mandatory to specify determinants for query subjects that represent a single level or fact data.

Cognos 8.1 or 8.2 Model Upgrade

Upgrading a model prepares the model to take advantage of the new features in Cognos 8.3.

To upgrade your model, follow these steps:

- Verify the model in your existing environment (p. 379).
- Open and upgrade the model (p. 379).
- Verify and repair the model (p. 379).

Verifying the Model in Cognos 8.1 or 8.2

The upgrade process does not resolve any problems that existed in your model prior to upgrading. Before upgrading the model, we recommend that you verify your model and fix any reported problems in your existing environment.

When upgrading from Cognos 8.1 or 8.2 to Cognos 8.3, all you need to do is verify the model and fix errors.

Opening and Upgrading the Cognos 8.1 or 8.2 Model

When you open an older model in Framework Manager, you are notified that the model must be updated.

When upgrading from Cognos 8.1 or 8.2 to Cognos 8.3, you should understand the impact of governors that are set during upgrade (p. 379).

If you are upgrading a model that is segmented or linked, see "Upgrading Segmented and Linked Projects" (p. 380).

Governors for a Cognos 8.1 or 8.2 Upgrade

Several governors are set automatically during the upgrade process:

- The **Shortcut Processing** governor is set to **Automatic**.

  This governor controls how shortcuts are processed by Cognos 8. When the governor is set to **Automatic**, the model works the same as in ReportNet, that is, a shortcut that exists in the same folder as its target behaves as an alias, or independent instance. However, a shortcut existing elsewhere in the model behaves as a reference to the original. When you create a new model, the **Shortcut Processing** governor is always set to **Explicit**.

- The **Suppress Null values for SAP BW data sources** governor is selected for SAP BW metadata only.

  This controls the creation of outer joins in models based on SAP BW metadata.

  Some queries can be very large because null values are not filtered out. Null suppression removes a row or column for which all of the values in the row or column are null (empty). Null suppression is performed by SAP BW. This reduces the amount of data transferred to the Cognos 8 studios and improves performance.

Verifying and Repairing the Upgraded Cognos 8.1 or 8.2 Model

After updating the metadata, Framework Manager prompts you to verify the model. We recommend that you verify one namespace at a time in large models. You can also verify an individual object.

When upgrading from Cognos 8.1 or 8.2 to Cognos 8.3, you need to select and repair objects.

The repair process first evaluates all selected items. This evaluation automatically resolves issues with new data types and prompts you to repair dimension information in the model.

You can select one or more check boxes of items and repair them.

Upgrading Segmented and Linked Projects

This information applies regardless of which version you are upgrading from.

A segmented project must be upgraded starting with the child segments at the lowest level of the hierarchy and proceeding toward the root, or master, project.

A linked project must be upgraded before upgrading any projects that use the linked project.

When upgrading segmented projects, we recommend that you follow this process.

- Open each segment as a separate project, starting with the lowest level segment in the hierarchy.
- Follow the steps to upgrade the model "Cognos ReportNet Model Upgrade" (p. 371) or "Cognos 8.1 or 8.2 Model Upgrade" (p. 378). Do not repair the segment.
- After the upgrade is complete, save the project.
- Upgrade each segment in the hierarchy, working back to the master project.
- Repair the complete model in the master project.
- After upgrading the master project, check in each segment and then check in the master project.

Appendix A: Troubleshooting

You may encounter problems when working in Framework Manager. For other troubleshooting topics, see the Cognos 8 *Administration and Security Guide*.

Measures Not Added After Initial Import in an Upgraded Model

After upgrading a Cognos ReportNet ® 1.x model to Cognos 8, measures that were added to the key figures folder for the underlying SAP BW object after the initial import will not be added when a synchronization is performed.

The solution is to re-import the object.

Note: This issue does not affect new models created in Cognos 8.

Unable to View the Result Set of a Stored Procedure

If the result set of a stored procedure contains a CLOB (character large object) or BLOB (binary large object), you must modify the stored procedure before running it.

Steps to Use CLOBS

1. Create a model query subject that references the stored procedure.
2. Insert a Cast or Substring function to calculate a string based on the CLOB.

Steps to Use BLOBS

1. Create a model query subject that references the stored procedure.
2. Remove the BLOB from the `Select` statement of the query subject.
3. Add query items to the `Select` statement that perform a lookup on the table that contains the BLOB data.

Unable to Compare Two CLOBs in Oracle

If you are using Oracle and ask Cognos 8 to compare two CLOBs, such as where C2 = C3, you will see an Oracle runtime error.

To avoid this problem, use the DBMS_LOB.compare method:

```
where 0 = dmbs_lob.compare (c1, c2)
```

An Out of Memory Error with ERWin Imported Metadata

When you test query subjects based on a View table, an Out of Memory error may occur while performing the `sqlPrepareWithOptions` operation.

The solution is to create a data source query subject using the same Content Manager connection as the ERWin model.

Calculation Fails Testing

You import a user-defined function, such as CHAR, that uses the same name as a vendor-specific function, such as char in Microsoft SQL Server. You insert the user-defined function in a calculation and test it in the **Calculation Definition** dialog box. The test fails.

The solution is not to use the same name as a vendor-specific function when creating a user-defined function. For information about function names used in your database, see the database product documentation.

Framework Manager Cannot Access the Gateway URI

You create a new project in Framework Manager and the following message appears:

Unable to access service at URL:

http://hostname:80/cognos8/cgi-bin/cognos.cgi/

b_acton=xts.run&m=portal/close.xts

Please check that your gateway URI information is configured correctly and that the service is available.

For further information please contact your service administrator.

This message appears if the gateway is not properly configured. The gateway URI must be set to the computer name where Cognos 8 is installed and reflect the type of gateway you are using. You must log on as an administrator to configure the gateway URI.

Steps to Configure the Gateway URI

1. Close Framework Manager.
2. In Cognos Configuration, in the **Explorer** window, click **Environment**.
3. In the **Properties** window, in the **Gateway URI** box, type the appropriate value:
   - To use ISAPI, replace cognos.cgi with **cognosisapi.dll**.
   - To use apache_mod, replace cognos.cgi with **mod_cognos.dll**.
   - To use a servlet gateway, type the following:

     http[s]://host:port/context_name/servlet/Gateway

Note:*context_name* is the name you assigned to the ServletGateway Web application when you deployed the ServletGateway WAR file.

- If you are not using a Web server, to use the dispatcher as the gateway, type the following:

  http[s]://*host***:***port***/p2pd/servlet/dispatch**

4. If required, change the host name portion of the **Gateway URI** from localhost to either the IP address of the computer or the computer name.
5. From the **File** menu, click **Save**.
6. From the **Actions** menu, click **Restart**.

Object Names Appear in the Wrong Language

When you import multiple languages from an SAP BW Query to a Framework Manager model, not all the object names retrieved from SAP BW appear in the correct language.

To avoid this problem, save the SAP BW Query again in each of the logon languages in Business Explorer Query Designer. The correct language texts will then show correctly in Framework Manager.

Full Outer Joins in Oracle Return Incorrect Results

When using an Oracle data source prior to version 10.2, full outer joins return incorrect data results. To avoid this problem, Cognos 8 processes these as local operations.

As a result of this processing, you must set the query processing to limitedLocal for any projects that expect explicit or implicit outer joins.

Error When Testing Query Subjects in a Model Imported from Teradata

You are using a model imported from Teradata. When you test some query subjects that contain graphic items, you see this error:

QE-DEF-0177 An error occurred while performing operation 'sqlOpenResult' status='-28'.

UDA-SQL-0114 The cursor supplied to the operation "sqlOpenResult" is inactive.

UDA-SQL-0107 A general exception has occurred during the operation "SgiCursor::doOpenResult ()".

[NCR][ODBC Teradata Driver][Teradata RDBMS] An unknown character string translation was requested.

The reason is that the `GRAPHIC` and `VARGRAPHIC` data types are not supported.

Error for Type-In SQL Query Subject

You define the following in DB2:

```
create type address as (
 number character (6),
 street varchar(35),
 city varchar(35)
 )
MODE DB2SQL;
create table emp ( emp_no int, emp_address address);
Select e.emp_no, e.emp_address..street from emp e
SQL0206N "aBmtQuerySubject.2
```

When you define a type-in SQL query subject, an error appears because of the name assigned for the attribute reference in the structured type.

To resolve this problem, you have two options:

- assign a simple correlation name to the column in the original query subject, such as `Select e.emp_no, e.emp_address..street as "ABC" from emp e`
- use pass-through notation for the query subject by surrounding the column with double curly brackets ({{ }})

Function Name Not Recognized

Cognos 8 does not validate the syntax you enter in expressions. The expression parser does not check the number or types of arguments for any function you use. This is because the database may accept different syntax or may have a function with the same name as Cognos 8 but a different signature.

If an expression will not run, review the **Tips** box in the expression editor for more information about the specific function you are using. The **Tips** box gives detailed information about each function.

QE-DEF-0259 Error

This error occurs if you use braces { } in the wrong position in an expression. Cognos 8 expects anything between the braces { } to be at the same level as a function. If you have used braces elsewhere in an expression, you will see the following error message:

QE-DEF-0259 There was a parsing error

You can also use braces to send unique syntax to the data source. For example, your database uses a keyword for a function but this keyword is not used in Cognos 8.

Cognos 8 does not validate the syntax you enter between braces. The syntax is simply sent to the data source.

The solution is to make sure that braces are not used in the wrong positions.

For example, you type the following in an expression:

```
[ss_ole_both].[authors_lith].[au_id] = [ss_ole_both].[authors_latin].[au_id]
 { collate Lithuanian_CI_AI}
```

You see the following error message:

```
QE-DEF-0259 There was a parsing error before or near
position: 75, text starting at position: 5 "le_both].[authors_lith].[au_id]=
[ss_ole_both].[authors_latin].[au_id]{"
```

Meanwhile the following expression is valid:

```
{ Q3.au_id }  =  { Q4.au_id collate lithuanian_CI_AI
}
```

QE-DEF-0260 Parsing Error

This error occurs if you pick a list of Query Studio filters that is too large or when the size of the string values for the keys is large.

One solution is to use predefined filters from Framework Manager instead of using the Query Studio interface. Also, we recommend that you pick columns and filter where the distinct list of values is relatively small.

Externalized Key Figures Dimension Retains Old Prompt Value

You have a key figures dimension (SAP BW) that contains an optional prompt. If you externalize this dimension as a csv or tab file, the externalized file does not contain all the rows of data. This is because the prompt value is retained.

For example, you set the prompt value for the dimension when testing the dimension in Framework Manager. The prompt value is kept in the cache. Even if you clear the value of the prompt in the Prompt dialog box, externalizing the key figures dimension results in a file containing data that is filtered by the most recently used prompt.

To avoid this problem, do one of the following:

- Do not test the key figures dimension before you externalize it.
- Close the model, open it again, and externalize the key figures dimension.

Older Cognos 8 Models Display Level Object Security

If you are using a previously-created Cognos 8 model, object security on a levelmay have been defined. Object security on a level is not supported.

The solution is to verify and repair the older model before publishing it.

Steps

1. From the **Project** menu, click **Verify Model**.
2. Select the security view that references a level and click **Repair**.

Exporting a Framework Manager Model to a CWM File Fails With Error *MILOG.TXT was not found*

Exporting a Framework model to a CWM file fails with error *MILOG.TXT was not found* when the path contains Japanese characters.

Do one of the following to solve this problem:

- Specify an export path that does not use Japanese characters.
- Change the system default language on your computer to Japanese. You can set the system default language in the Control Panel, under **Regional and Language Options** -> **Advanced**. For more information on how to do this, refer to the Windows operating system help.

Difference in SQL for Inner Joins After Upgrading to Cognos 8.3

If you migrated from Cognos 8.1 to Cognos 8.2 or 8.3, there can be differences in the generation of SQL used for the `INNER JOIN` syntax. You can control the SQL syntax used for inner joins by selecting the **Server determined** setting for the **SQL Join Syntax** governor. The SQL generated in Cognos 8.1, Cognos 8.2, and Cognos 8.3 produces the same results.

If you are using RDBMS materialization technology which can be implemented using either implicit or explicit syntax, you must ensure that you select the same syntax setting for the **SQL Join Syntax** governor in your model.

Unable to Select a Command for a Query Item in a Shortcut

You have a shortcut to a query subject or dimension. A shortcut is a pointer to a source object and the items in a shortcut are also shortcuts.

You cannot use shortcut query items in commands such as **Create**, **Relationship**.

Publishing a Large Package Results in a CCLOutOfMemory Error

When you try to publish a large package, you see this error message:

CCLOutOfMemory

To resolve this problem, reduce the size of your package by removing namespaces. You could also create a series of smaller packages that each target specific business goals.

Note: Publishing very large packages may negatively affect the future performance and maintainability of your models.

Appendix B: Using the Expression Editor

An expression is any combination of operators, constants, functions, and other components that evaluates to a single value. You build expressions to create calculation and filter definitions. A calculation is an expression that you use to create a new value from existing values contained within a data item. A filter is an expression that you use to retrieve a specific subset of records.

The expression editor shows the expression components that are supported by the data source in which the metadata is stored. For example, if you import metadata from an Oracle data source, the expression editor shows only the elements that are supported in Oracle.

If you are using a DB2 data source, note that the subtract operator is invalid if you combine the datatypes `timestamp2` and `packed decimal`.

When creating an expression that will be used in a double-byte environment, such as Japanese, the only special characters that will work are ASCII-7 and ~ -- ‖ - $ ¢ £ ¬.

Quality of Service Indicators

Not all data sources support functions the same way. The data modeler can set a quality of service indicator on functions to give a visual clue about the behavior of the functions. Report authors can use the quality of service indicators to determine which functions to use in a report. The quality of service indicators are:

- not available (X)

  This function is not available for any data source in the package.

- limited availability (!!)

  The function is not available for some data sources in the package.

- poor performance (!)

  The function is available for all data sources in the package but may have poor performance in some data sources.

- unconstrained (no symbol)

  The function is available for all data sources.

SAP BW Support

SAP BW does not support all operators or summaries. This can be confusing if you have imported SAP BW metadata and non-SAP BW metadata into the same model.

SAP BW does not support the following operators:

- like
- lookup

SAP BW does not support the following member summaries:

- date-time
- interval
- interval month
- interval day
- interval day to hour
- interval day to minute
- interval day to second
- interval hour
- interval hour to minute
- interval hour to second
- interval minute
- interval minute to second
- interval second
- interval year
- interval year to month
- moving
- running
- time with time zone
- timestamp with time zone

Cell values are date, number, or time. Attribute values are strings.

Searching for Values May Return Unexpected Results

In the expression editor, when searching for values for a data item, the results you obtain may contain unexpected results if the data item is not a string data type. Because users can edit the expression for a data item, Cognos 8 cannot determine with certainty what the data type is.

Therefore, Cognos 8 guesses the data type of the data item by looking at its aggregate and roll-up aggregate set.

Operators

Operators specify what happens to the values on either side of the operator. Operators are similar to functions, in that they manipulate data items and return a result.

(

Inserts an open parenthesis in your expression.

Syntax

```
(expression)
```

)

Inserts a close parenthesis in your expression.

Syntax

```
(expression)
```

*

Multiplies two numeric values.

Syntax

```
value1 * value2
```

,

Separates expression components.

Syntax

```
expression(param1, param2)
```

/

Divides two numeric values.

Syntax

```
value1 / value2
```

||

Concatenates strings.

Syntax

```
string1 || string2
```

+

Adds two values.

Syntax

```
value1 + value2
```

-

Subtracts two numeric values or negates a numeric value.

Syntax

```
value1 - value2
or
- value
```

<

Compares values against a defined value and retrieves the values that are less than the defined value.

Syntax

```
value1 < value2
```

<=

Compares values against a defined value and retrieves the values that are less than or equal to the defined value.

Syntax

```
value1 <= value2
```

<>

Compares values against a defined value and retrieves the values that are not equal to the defined value.

Syntax

```
value1 <> value2
```

=

Compares values against a defined value and retrieves the values that are equal to the defined value.

Syntax

```
value1 = value2
```

>

Compares values against a defined value and retrieves the values that are greater than the defined value.

Syntax

```
value1 > value2
```

->

Used as a separator in a literal member expression.

Syntax

```
[namespace].[dimension].[hierarchy].[level]->[L1]
```

>=

Compares values against a defined value and retrieves the values that are greater than or equal to the defined value.

Syntax

```
value1 >= value2
```

and

Returns true if the conditions on both sides of the expression are true.

Syntax

```
arg1 AND arg2
```

auto

Use with summary expressions to define the scope to be adjusted based on the grouping columns in the query. The scope is context dependent.

Syntax

```
aggregate_function ( expression AUTO )
```

between

Determines if a value falls in a given range.

Syntax

```
name BETWEEN value1 and value2
```

case

Use with When, Then, Else, and End.

Syntax

```
CASE expression { WHEN expression THEN expression } [ ELSE expression ] END
```

contains

Determines if a string contains another string.

Syntax

```
string1 CONTAINS string2
```

currentMeasure

Keyword that can be used as the first argument of member summary functions.

Syntax

```
aggregate_function( currentMeasure within set expression )
```

default

Use with LOOKUP construct.

Syntax

```
lookup(....) in (....) DEFAULT(....)
```

distinct

A keyword used in an aggregate expression, to include only distinct occurrences of values. See also the function unique.

Syntax

```
DISTINCT dataItem
```

Example

```
count ( DISTINCT [OrderDetailQuantity] )
```

Result: 1704

else

Use with If or Case constructs.

Syntax

```
IF (condition) THEN .... ELSE (expression) , or CASE .... ELSE expression END
```

end

Use with Case When construct.

Syntax

```
CASE .... END
```

ends with

Determines if a string ends with a given string.

Syntax

```
string1 ENDS WITH string2
```

for

Use with summary expressions to define the scope of the aggregation in the query.

Syntax

```
aggregate_function ( expression FOR expression { , expression } )
```

for all

Use with summary expressions to define the scope to be all the specified grouping columns in the query. See also FOR clause.

Syntax

```
aggregate_function ( expression FOR ALL expression { , expression } )
```

for any

Use with summary expressions to define the scope to be adjusted based on a subset of the grouping columns in the query. Equivalent to FOR clause.

Syntax

```
aggregate_function ( expression FOR ANY expression { , expression } )
```

for report

Use with summary expressions to define the scope to be the whole query. See also FOR clause.

Syntax

```
aggregate_function ( expression FOR REPORT )
```

if

Use with Then and Else.

Syntax

```
IF (condition is true) THEN (action) ELSE (alternate action)
```

in

Determines if a value exists in a given list of values.

Syntax

```
exp1 IN (exp_list)
```

in_range

Determines if an item exists in a given list of constant values or ranges.

Syntax

```
expression IN_RANGE { constant : constant [ , constant : constant ] }
```

is missing

Determines if a value is undefined in the data.

Syntax

```
value IS MISSING
```

is null

Determines if a value is undefined in the data.

Syntax

```
value IS NULL
```

is not missing

Determines if a value is defined in the data.

Syntax

```
value IS NOT MISSING
```

is not null

Determines if a value is defined in the data.

Syntax

```
value IS NOT NULL
```

like

Determines if a string matches the pattern of another string.

Syntax

```
string1 LIKE string2
```

lookup

Finds and replaces data with a value you specify. It is preferable to use the CASE construct.

Syntax

```
LOOKUP (name) in (value1 --> value2) default (expression)
```

Example

```
lookup([Country]) in ('Canada'-->([List Price] * 0.60), 'Australia'-->([List
Price] * 0.80)) default([List Price])
```

not

Returns true if the condition is false, otherwise returns false.

Syntax

```
NOT arg
```

or

Returns true if either of the two conditions on both sides of the expression is true.

Syntax

```
arg1 OR arg2
```

prefilter

Performs a summary calculation before applying the summary filter.

Syntax

```
summary ([expression] PREFILTER)
```

rows

Counts the number of rows output by the query. Use with Count().

Syntax

```
count(ROWS)
```

starts with

Determines if a string starts with a given string.

Syntax

```
string1 STARTS WITH string2
```

then

Use with If or Case constructs.

Syntax

```
IF (condition) THEN ...., or CASE expression WHEN expression THEN .... END
```

when

Use with Case construct.

Syntax

```
CASE [expression] WHEN .... END
```

Summaries

This list contains predefined functions that return either a single summary value for a group of related values or a different summary value for each instance of a group of related values.

aggregate

Returns a calculated value using the appropriate aggregation function, based on the aggregation type of the expression.

Syntax

```
aggregate ( expr [ auto ] )
aggregate ( expr for [ all | any ] expr { , expr } )
aggregate ( expr for report )
```

average

Returns the average value of selected data items. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
average ( [ distinct ] expr [ auto ] )
average ( [ distinct ] expr for [ all | any ] expr { , expr } )
average ( [ distinct ] expr for report )
```

Example

```
average ( Sales )
```

Result: The average of all Sales values.

count

Returns the number of selected data items excluding NULL values. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
count ( [ distinct ] expr [ auto ] )
count ( [ distinct ] expr for [ all | any ] expr { , expr } )
count ( [ distinct ] expr for report )
```

Example

```
count ( Sales )
```

Result: The total number of entries under Sales.

maximum

Returns the maximum value of selected data items. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
maximum ( [ distinct ] expr [ auto ] )
maximum ( [ distinct ] expr for [ all | any ] expr { , expr } )
maximum ( [ distinct ] expr for report )
```

Example

```
maximum ( Sales )
```

Result: The maximum value of all Sales values.

median

Returns the median value of selected data items.

Syntax

```
median ( expr [ auto ] )
median ( expr for [ all | any ] expr { , expr } )
median ( expr for report )
```

minimum

Returns the minimum value of selected data items. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
minimum ( [ distinct ] expr [ auto ] )
minimum ( [ distinct ] expr for [ all | any ] expr { , expr } )
minimum ( [ distinct ] expr for report )
```

Example

```
minimum ( Sales )
```

Result: The minimum value of all Sales values.

moving-average

Returns a moving average by row for a specified set of values of over a specified number of rows. The <for-option> defines the scope of the function. The at option defines the level of aggregation and can only be used in the context of relational datasources. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
moving-average ( numeric_expr , numeric_expr  [ at exp {, expr } ]  [
<for-option> ] [ prefilter ] )
moving-average ( [ distinct ] numeric_expr , numeric_expr [ <for-option> ] [
prefilter ] )
<for-option> ::= for expr {, expr } | for report | auto
```

Example

```
moving-average ( Qty, 3 )
```

Result: For each row, this displays the quantity and a moving average of the current row and the preceding two rows.

```
Qty               Moving-Average (Qty, 3)
------            -----------------------------------
200               200
700               450
400               433.3333
200               433.3333
200               266.6667
500               300.0000
```

moving-total

Returns a moving total by row for a specified set of values over a specified number of rows. The <for-option> defines the scope of the function. The at option defines the level of aggregation and can only be used in the context of relational datasources. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
moving-total ( numeric_expr , numeric_expr  [ at exp {, expr } ]  [ <for-option>
 ] [ prefilter ] )
moving-total ( [ distinct ] numeric_expr , numeric_expr [ <for-option> ] [
prefilter ] )
<for-option> ::= for expr {, expr } | for report | auto
```

Example

```
moving-total ( Quantity, 3 )
```

Result: For each row, this displays the quantity and a moving total of the current row and the preceding two rows.

```
Qty               Moving-Total (Qty, 3)
------            ---------------------------------
200               200
700               900
400               1300
200               1300
200               800
500               900
```

percentage

Returns the percent of the total value for selected data items. The <for-option> defines the scope of the function. The at option defines the level of aggregation and can only be used in the context

of relational datasources. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
percentage ( numeric_expr  [ at exp {, expr } ]  [ <for-option> ] [ prefilter
] )
percentage ( [ distinct ] numeric_expr [ <for-option> ] [ prefilter ] )
<for-option> ::= for expr {, expr } | for report | auto
```

Example

```
percentage ( sales 98 )
```

Result: Shows the percentage of the total sales for 1998 that is attributed to each sales representative.

Sales Rep	Sales 98	Percentage
Bill Gibbons	60646	7.11%
Bjorn Flertjan	62523	7.35%
Chris Cornel	22396	2.63%

percentile

Returns a value, on a scale of one hundred, that indicates the percent of a distribution that is equal to or below the selected data items. The <for-option> defines the scope of the function. The at option defines the level of aggregation and can only be used in the context of relational datasources. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
percentile ( numeric_expr  [ at exp {, expr } ] [ <for-option> ] [ prefilter ]
 )
percentile ( [ distinct ] numeric_expr [ <for-option> ] [ prefilter ] )
<for-option> ::= for expr {, expr } | for report | auto
```

Example

```
percentile ( Sales 98 )
```

Result: For each row, displays the percentage of rows that are equal to or less than the quantity value of that row.

```
Qty               Percentile (Qty)
------            -----------------
800               1
700               0.875
600               0.75
500               0.625
400               0.5
400               0.5
200               0.25
200               0.25
```

quantile

Returns the rank of a value for a range that you specify. It returns integers to represent any range of ranks, such as 1 (highest) to 100 (lowest). The <for-option> defines the scope of the function. The at option defines the level of aggregation and can only be used in the context of relational datasources. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
quantile ( numeric_expr, numeric_expr  [ at exp {, expr } ]  [ <for-option> ]
[ prefilter ] )
quantile ( [ distinct ] numeric_expr, numeric_expr  [ <for-option> ] [ prefilter
 ] )
<for-option> ::= for expr {, expr } | for report | auto
```

Example

```
quantile ( Qty, 4 )
```

Result: The quantity, the rank of the quantity value, and the quantity values broken down into 4 quantile groups (quartiles).

```
Qty         Rank (Qty)  Quantile (Qty, 4)
------      ----------- ----------------------
800         1           1
700         2           1
600         3           2
500         4           2
400         5           3
400         5           3
200         7           4
200         7           4
```

quartile

Returns the rank of a value, represented as integers from 1 (highest) to 4 (lowest), relative to a group of values. The <for-option> defines the scope of the function. The at option defines the level of aggregation and can only be used in the context of relational datasources. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
quartile ( numeric_expr  [ at exp {, expr } ]  [ <for-option> ] [ prefilter ]
)
quartile ( [ distinct ] numeric_expr [ <for-option> ] [ prefilter ] )
<for-option> ::= for expr {, expr } | for report | auto
```

Example

```
quartile ( Qty )
```

Result: This displays the quantity and the quartile of the quantity value represented as integers from 1 (highest) to 4 (lowest).

Qty	Quartile (Qty)
450	1
400	1
350	2
300	2
250	3
200	3
150	4
100	4

rank

Returns the rank value of selected data items. The sort order is optional; descending order (DESC) is assumed by default. If two or more rows tie, then there is a gap in the sequence of ranked values (also known as Olympic ranking). The <for-option> defines the scope of the function. The at option defines the level of aggregation and can only be used in the context of relational datasources. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product. Null values are ranked last.

Syntax

```
rank ( expr [ ASC | DESC ] {, expr [ ASC | DESC ] }  [ at exp {, expr } ]  [
<for-option> ] [ prefilter ] )
rank ( [ distinct ] expr [ ASC | DESC ] {, expr [ ASC | DESC ] } [ <for-option>]
```

```
  [ prefilter ] )
<for-option> ::= for expr {, expr } | for report | auto
```

Example

```
rank ( Sales 98 )
```

Result: For each row, this displays the rank value of sales for 1998 that is attributed to each sales representative, and skips some numbers when there is a tie between rows.

Sales Rep	Sales 98	Rank
Bill Gibbons	60000	1
Bjorn Flertjan	50000	2
Chris Cornel	50000	2
John Smith	48000	4

running-average

Returns the running average by row (including the current row) for a set of values. The <for-option> defines the scope of the function. The at option defines the level of aggregation and can only be used in the context of relational datasources. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
running-average ( numeric_expr  [ at exp {, expr } ]  [ <for-option> ] [
prefilter ] )
running-average ( [ distinct ] numeric_expr [ <for-option> ] [ prefilter ] )
<for-option> ::= for expr {, expr } | for report | auto
```

Example

```
running-average ( Qty )
```

Result: For each row, this displays the quantity and a running average of the current and the previous rows.

Name	Qty	Avg	Running-Average for name
Smith	7	5	7
Smith	3	5	5
Smith	6	5	5.33
Smith	4	5	5
Wong	3	4	3
Wong	5	4	4

running-count

Returns the running count by row (including the current row) for a set of values. The <for-option> defines the scope of the function. The at option defines the level of aggregation and can only be used in the context of relational datasources. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
running-count ( numeric_expr  [ at exp {, expr } ]  [ <for-option> ] [ prefilter
 ] )
running-count ( [ distinct ] numeric_expr  [ <for-option> ] [ prefilter ] )
<for-option> ::= for expr {, expr } | for report | auto
```

Example

```
running-count ( Qty )
```

Result: For each row, this displays the quantity and a running count of the position of current row.

Name	Qty	Count	Running-Count for name
Smith	7	4	1
Smith	3	4	2
Smith	6	4	3
Smith	4	4	4
Wong	3	3	1
Wong	5	3	2

running-difference

Returns a running difference by row, calculated as the difference between the value for the current row and the preceding row, (including the current row) for a set of values. The <for-option> defines the scope of the function. The at option defines the level of aggregation and can only be used in the context of relational datasources. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
running-difference ( numeric_expr [ at exp {, expr } ]  [ <for-option> ] [
prefilter ] )
running-difference ( [ distinct ] numeric_expr  [ <for-option> ] [ prefilter ]
 )
<for-option> ::= for expr {, expr } | for report | auto
```

Example

```
running-difference ( Qty )
```

Result: For each row, this displays the quantity and a running difference between the value for the current row and the preceding row.

```
Name        Qty         Running-Difference for name
-------     ------      ---------------------------------------
Smith       7           NULL
Smith       3           -4
Smith       6           3
Smith       4           -2
Wong        3           -1
Wong        5           2
```

running-maximum

Returns the running maximum by row (including the current row) for a set of values. The <for-option> defines the scope of the function. The at option defines the level of aggregation and can only be used in the context of relational datasources. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
running-maximum (  numeric_expr  [ at exp {, expr } ]  [ <for-option> ] [
prefilter ] )
running-maximum ( [ distinct ] numeric_expr  [ <for-option> ] [ prefilter ] )
<for-option> ::= for expr {, expr } | for report | auto
```

Example

```
running-maximum ( Qty )
```

Result: For each row, this displays the quantity and a running maximum of the current and previous rows.

```
Name        Qty         Max         Running-Maximum (Qty) for name
-------     ------      ------      ------------------------------------
Smith       2           7           2
Smith       3           7           3
Smith       6           7           6
Smith       7           7           7
Wong        3           5           3
Wong        5           5           5
```

running-minimum

Returns the running minimum by row (including the current row) for a set of values. The <for-option> defines the scope of the function. The at option defines the level of aggregation and can only be used in the context of relational datasources. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
running-minimum ( numeric_expr  [ at exp {, expr } ]  [ <for-option> ] [
prefilter ] )
running-minimum ( [ distinct ] numeric_expr  [ <for-option> ] [ prefilter ] )
<for-option> ::= for expr {, expr } | for report | auto
```

Example

```
running-minimum ( Qty )
```

Result: For each row, this displays the quantity and a running minimum of the current and previous rows.

Name	Qty	Min	Running-Minimum (Qty) for name
Smith	7	2	7
Smith	3	2	3
Smith	6	2	3
Smith	2	2	2
Wong	4	3	4
Wong	5	3	4

running-total

Returns a running total by row (including the current row) for a set of values. The <for-option> defines the scope of the function. The at option defines the level of aggregation and can only be used in the context of relational datasources. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
running-total ( numeric_expr [ at exp {, expr } ]   [ <for-option> ] [ prefilter
 ] )
running-total ( [ distinct ] numeric_expr  [ <for-option> ] [ prefilter ] )
<for-option> ::= for expr {, expr } | for report | auto
```

Example

```
running-total ( Qty )
```

Result: For each row, this displays the quantity and a running total of the current and previous rows.

Name	Qty	Total	Running-Total (Qty) for name
Smith	2	18	2
Smith	3	18	5
Smith	6	18	11
Smith	7	18	18
Wong	3	12	3
Wong	5	12	8

standard-deviation

Returns the standard deviation of selected data items. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
standard-deviation ( [ distinct ] expr [ auto ] )
standard-deviation ( [ distinct ] expr for [ all | any ] expr { , expr } )
standard-deviation ( [ distinct ] expr for report )
```

Example

```
standard-deviation ( ProductCost )
```

Result: A value indicating the deviation between product costs and the average product cost.

standard-deviation-pop

Computes the population standard deviation and returns the square root of the population variance. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
standard-deviation-pop ( [ distinct ] expr [ auto ] )
standard-deviation-pop ( [ distinct ] expr for [ all | any ] expr { , expr } )

standard-deviation-pop ( [ distinct ] expr for report )
```

Example

```
standard-deviation-pop ( ProductCost )
```

Result: A value of the square root of the population variance.

total

Returns the total value of selected data items. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
total ( [ distinct ] expr [ auto ] )
total ( [ distinct ] expr for [ all | any ] expr { , expr } )
total ( [ distinct ] expr for report )
```

Example

```
total ( Sales )
```

Result: The total value of all Sales values.

variance

Returns the variance of selected data items. The keyword distinct is available for backward compatibility of expressions used in previous versions of the product.

Syntax

```
variance ( [ distinct ] expr [ auto ] )
variance ( [ distinct ] expr for [ all | any ] expr { , expr } )
variance ( [ distinct ] expr for report )
```

Example

```
variance ( Product Cost )
```

Result: A value indicating how widely product costs vary from the average product cost.

variance-pop

Returns the population variance of a set of numbers after discarding the nulls in this set.

Syntax

```
variance-pop ( [ distinct ] expr [ auto ] )
variance-pop ( [ distinct ] expr for [ all | any ] expr { , expr } )
variance-pop ( [ distinct ] expr for report )
```

Example

```
variance-pop ( Qty)
```

Result: For each row, this displays the population variance of a set of numbers after discarding the nulls in this set.

Member Summaries

This list contains predefined functions that return either a single summary value for a set of members or a different summary value for each member of a set of members.

aggregate

Returns a calculated value using the appropriate aggregation function, based on the aggregation type of the expression.

Syntax

```
aggregate ( < currentMeasure | expr > within set set_expr { , set_expr } )
aggregate ( < currentMeasure | expr > within < detail | aggregate > expr )
```

average

Returns the average value of selected data items.

Syntax

```
average ( < currentMeasure | expr > within set set_expr { , set_expr } )
average ( < currentMeasure | expr > within < detail | aggregate > expr )
```

count

Returns the number of selected data items excluding NULL values.

Syntax

```
count ( < currentMeasure | expr > within set set_expr { , set_expr } )
count ( < currentMeasure | expr > within < detail | aggregate > expr )
```

maximum

Returns the maximum value of selected data items.

Syntax

```
maximum ( < currentMeasure | expr > within set set_expr { , set_expr } )
maximum ( < currentMeasure | expr > within < detail | aggregate > expr )
```

median

Returns the median value of selected data items.

Syntax

```
median ( < currentMeasure | expr > within set set_expr { , set_expr } )
median ( < currentMeasure | expr > within < detail | aggregate > expr )
```

minimum

Returns the minimum value of selected data items.

Syntax

```
minimum ( < currentMeasure | expr > within set set_expr { , set_expr } )
minimum ( < currentMeasure | expr > within < detail | aggregate > expr )
```

percentage

Returns the percent of the total value for selected data items.

Syntax

```
percentage ( numeric_expr  [ tuple member_expr {, member_expr } ] within set
set_expr  {, set_expr } )
```

Example

```
percentage ( [gosales].[sales measures].[quantity] tuple [gosales].[Staff].[].
[department]->[West] within set children([gosales].[Staff].[].[Staff] )
```

percentile

Returns a value, on a scale of one hundred, that indicates the percent of a distribution that is equal to or below the selected data items.

Syntax

```
percentile ( numeric_expr  [ tuple member_expr {, member_expr } ] within set
set_expr  {, set_expr } )
```

quantile

Returns the rank of a value for the specified range. It returns integers to represent any range of ranks, such as 1 (highest) to 100 (lowest).

Syntax

```
quantile ( numeric_expr, numeric_expr  [ tuple member_expr {, member_expr } ]
within set set_expr  {, set_expr } )
```

quartile

Returns the rank of a value, represented as integers from 1 (highest) to 4 (lowest), relative to a group of values.

Syntax

```
quartile ( numeric_expr [ tuple member_expr {, member_expr } ] within set
set_expr  {, set_expr } )
```

rank

Returns the rank value of selected data items. The type of ranking returned (Olympic, dense or serial) is data source dependent. The sort order is optional; DESC is assumed by default.

Syntax

```
rank ( numeric_expr [ ASC | DESC ] [ tuple member_expr {, member_expr } ] within
 set set_expr  {, set_expr } )
```

Example

```
rank  ( [gosales].[sales measures].[quantity] tuple [gosales].[Staff].[].
[department]->[West] within set children([gosales].[Staff].[].[Staff] )
```

standard-deviation

Returns the standard deviation of selected data items.

Syntax

```
standard-deviation ( < currentMeasure | expr > within set set_expr { , set_expr
 } )
standard-deviation ( < currentMeasure | expr > within < detail | aggregate >
expr )
```

total

Returns the total value of selected data items.

Syntax

```
total ( < currentMeasure | expr > within set set_expr { , set_expr } )
total ( < currentMeasure | expr > within < detail | aggregate > expr )
```

variance

Returns the variance of selected data items.

Syntax

```
variance ( < currentMeasure | expr > within set set_expr { , set_expr } )
variance ( < currentMeasure | expr > within < detail | aggregate > expr )
```

Constants

A constant is a fixed value that you can use in an expression.

date

Inserts the current system date.

date-time

Inserts the current system date and time.

time with time zone

Inserts a zero time with time zone.

timestamp with time zone

Inserts an example of a timestamp with time zone.

interval

Inserts a zero interval.

interval year

Inserts a zero year interval.

interval month

Inserts a zero month interval.

interval year to month

Inserts a zero year to month interval.

interval day

Inserts a zero day interval.

interval hour

Inserts a zero hour interval.

interval minute

Inserts a zero minute interval.

interval second

Inserts a zero second interval.

interval day to hour

Inserts a zero day to hour interval.

interval day to minute

Inserts a zero day to minute interval.

interval day to second

Inserts a zero day to second interval.

interval hour to minute

Inserts a zero hour to minute interval.

interval hour to second

Inserts a zero hour to second interval.

interval minute to second

Inserts a zero minute to second interval.

null

Inserts a null value if the expression conditions are not met.

number

Inserts the number 0, which you can replace with a new numeric value.

string

Inserts an empty string.

time

Inserts the current system time.

Constructs

A construct is a template that can be used to create an expression.

if then else

This is the template for using an "IF...THEN...ELSE" statement in the expression.

Syntax

```
IF ([Country] = 'Canada')
    THEN ([List Price] * 0.60)
    ELSE ([List Price])
```

in_range

This is the template for using "IN_RANGE" in the expression.

Syntax

```
example: [gosales].[CONVERSIONRATE].[COUNTRYCODE] IN_RANGE { :30 , 40, 50, 999:
 }
```

search case

This is the template for using a "search case" in the expression.

Syntax

```
CASE
    WHEN [Country] = 'Canada' THEN ([List Price] * 0.60)
    WHEN [CountryCode] > 100 THEN [List Price] * 0.80
    ELSE [List Price]
END
```

simple case

This is the template for using a "simple case" in the expression.

Syntax

```
CASE [Country]
    WHEN 'Canada' THEN ([List Price] * 0.60)
    WHEN 'Australia' THEN [List Price] * 0.80
    ELSE [List Price]
END
```

Business Date/Time Functions

This list contains business functions for performing date and time calculations.

_add_days

Returns the date or datetime dependent on the first argument resulting from adding integer_exp days to date_exp.

Syntax

```
_add_days ( date_exp, integer_exp )
```

Example 1

```
_add_days ( 2002-04-30 , 1 )
```

Result: 2002-05-01

Example 2

```
_add_days ( 2002-04-30 12:10:10.000, 1 )
```

Result: 2002-05-01 12:10:10.000

Example 3

```
_add_days ( 2002-04-30 00:00:00.000, 1/24 )
Note that the second argument is not a whole number. This is supported by some
 database technologies and increments the time portion.
```

Result: 2002-04-30 01:00:00.000

_add_months

Returns the date or datetime dependent on the first argument resulting from adding integer_exp months to date_exp.

Syntax

```
_add_months ( date_exp, integer_exp )
```

Example 1

```
_add_months ( 2002-04-30 , 1 )
```

Result: 2002-05-30

Example 2

```
_add_months ( 2002-04-30 12:10:10.000, 1 )
```

Result: 2002-05-30 12:10:10.000

_add_years

Returns the date or datetime dependent on the first argument resulting from adding integer_exp years to date_exp.

Syntax

```
_add_years ( date_exp, integer_exp )
```

Example 1

```
_add_years ( 2002-04-30 , 1 )
```

Result: 2003-04-30

Example 2

```
_add_years ( 2002-04-30 12:10:10.000 , 1 )
```

Result: 2003-04-30 12:10:10.000

_age

Returns a number that is obtained from subtracting date_exp from today's date. This value has the form YYYYMMDD, where YYYY represents the number of years, MM represents the number of months, and DD represents the number of days.

Syntax

```
_age (date_exp )
```

Example

```
Today's date=2003-02-05 _age ( 1990-04-30 )
```

Result: 120906 that is 12 years, 9 months and 6 days

_day_of_week

Returns the day of week (between 1 and 7), where 1 is the first day of the week as indicated by the second parameter(between 1 and 7, 1 being Monday and 7 being Sunday). Note that in ISO 8601 standard, a week begins with Monday being day 1. In North America where Sunday is the first day of the week being day 7.

Syntax

```
_day_of_week ( date_exp, integer )
```

Example

```
_day_of_week ( 2003-01-01, 1 )
```

Result: 3

_day_of_year

Returns the ordinal for the day of the year in date_ exp (1 to 366). Also known as Julian day.

Syntax

```
_day_of_year ( date_exp )
```

Example

```
_day_of_year ( 2003-03-01 )
```

Result: 61

_days_between

Returns a positive or negative number representing the number of days between the two date expressions. If date_exp1 < date_exp2 then the result will be a negative number.

Syntax

```
_days_between ( date_exp1, date_exp2 )
```

Example

```
_days_between ( 2002-04-30 , 2002-06-21 )
```

Result: -52

_days_to_end_of_month

Returns a number representing the number of days remaining in the month represented by the date expression date_exp.

Syntax

```
_days_to_end_of_month ( date_exp )
```

Example

```
_days_to_end_of_month ( 2002-04-20 14:30:22.123 )
```

Result: 10

_first_of_month

Returns a date or datetime dependent on the argument obtained from converting date_exp to a date with the same year and month but the day set to 1.

Syntax

```
_first_of_month ( date_exp )
```

Example 1

```
_first_of_month ( 2002-04-20 )
```

Result: 2002-04-01

Example 2

```
_first_of_month ( 2002-04-20 12:10:10.000 )
```

Result: 2002-04-01 12:10:10.000

_last_of_month

Returns a date or datetime dependent on the argument that is the last day of the month represented by date_exp.

Syntax

```
_last_of_month ( date_exp )
```

Example 1

```
_last_of_month ( 2002-01-14 )
```

Result: 2002-01-31

Example 2

```
_last_of_month ( 2002-01-14 12:10:10.000 )
```

Result: 2002-01-31 12:10:10.000

_make_timestamp

Returns a timestamp constructed from integer_exp1 (the year), integer_exp2 (the month) and integer_exp3 (the day). The time portion defaults to 00:00:00.000 .

Syntax

```
_make_timestamp ( integer_exp1, integer_exp2, integer_exp3 )
```

Example

```
_make_timestamp ( 2002 , 01 , 14 )
```

Result: 2002-01-14 00:00:00.000

_months_between

Returns a positive or negative integer number representing the number of months between date_exp1 to date_exp2. If date_exp1 < date_exp2, then a negative number is returned.

Syntax

```
_months_between ( date_exp1, date_exp2 )
```

Example

```
_months_between ( 2002-01-30, 2002-04-03 )
```

Result: 2

_week_of_year

Returns the number of the week of the year of the date_exp according to the ISO 8601 standard, in which week 1 of the year is the first week of the year to contain a Thursday, which is equivalent to the first week containing January 4th. A week starts on Monday (day 1) and ends on Sunday (day 7).

Syntax

```
_week_of_year ( date_exp )
```

Example

```
_week_of_year ( 2003-01-01 )
```

Result: 1

_years_between

Returns a positive or negative integer number representing the number of years from date_exp1 to date_exp2. If date_exp1 < date_exp2 then a negative value is returned.

Syntax

```
_years_between ( date_exp1, date_exp2 )
```

Example

```
_years_between ( 2003-01-30, 2001-04-03 )
```

Result: 1

_ymdint_between

Returns a number representing the difference between the date expressions date_exp1 and date_exp2. This value has the form YYYYMMDD, where YYYY represents the number of years, MM represents the number of months, and DD represents the number of days.

Syntax

```
_ymdint_between ( date_exp1, date_exp2 )
```

Example

```
_ymdint_between ( 1990-04-30 , 2003-02-05 )
```

Result: 120906 that is 12 years, 9 months and 6 days

Block Functions

This list contains functions used to access members of a set, usually in the context of Analysis Studio.

_firstFromSet

Returns the first members found in the set up to numeric_exp_max + numeric_exp_overflow. If numeric_exp_max + numeric_exp_overflow is exceeded, then only the max number of members are returned.

Syntax

```
_firstFromSet ( set_exp, numeric_exp_max , numeric_exp_overflow )
```

_remainderSet

The member expression will be included in the returned set when the size of the set_exp set is greater than numeric_exp.

Syntax

```
_remainderSet (member_exp,  set_exp , numeric_exp )
```

Macro Functions

This list contains functions that can be used within a macro. A macro may contain one or more macro functions. A macro is delimited by a number sign (#) at the beginning and at the end. Everything between the number signs is treated as a macro expression, which is executed at run time.

+

Concatenates two strings.

Syntax

```
value1 + value2
```

Example

```
# '{' + $runLocale + '}'#
```

Result: {en-us}

array

Constructs an array out of the list of parameters.

Syntax

```
array ( string_exp | array_exp { , string_exp | array_exp } )
```

Example

```
#csv ( 'x1' , 'x2' , array ( 'a1' , 'a2' ) )#
```

Result: 'x1' , 'x2' , 'a1' , 'a2'

csv

Constructs a comma separated values string from the elements of the array. Optionally the separator and quote strings can be specified. The default separator is a comma (,) and the default quote character is a single quote (').

Syntax

```
csv ( array_exp [ , separator_string [ , quote_string ] ] )
```

Example

```
#csv ( array ( 'a1' , 'a2' ) )#
```

Result: 'a1' , 'a2'

dq

Surround the passed string with double quotes.

Syntax

```
dq ( string_exp )
```

Example

```
#dq ( 'zero' )#
```

Result: "zero"

grep

Searches for elements of an array that match the pattern specified in the first argument. It returns an array with the elements that pass the pattern.

Syntax

```
grep ( pattern_string , array_exp )
```

Example

```
#csv ( grep ( 's' , array ( 'as', 'an', 'arts' ) ) )#
```

Result: 'as', 'arts'

join

Joins the elements of an array using the separator string.

Syntax

```
join ( separator_string , array_exp )
```

Example

```
# sq ( join ( ' | | ' , array ( 'as', 'an', 'arts' ) ) )#
```

Result: 'as | | an | | arts'

prompt

Prompt the user for a single value. Only the prompt_name argument is required. The datatype defaults to string when not specified. The prompt is optional when defaultText is specified. The text, when specified, will precede the value. A queryItem can be specified to take advantage of the Prompt Info properties of the query item. The trailing_text, when specified, will be appended to the value.

Syntax

```
prompt ( prompt_name , datatype , defaultText , text , queryItem , trailing_text
 )
```

Example

```
select . . . where COUNTRY_MULTILINGUAL.COUNTRY_CODE  > #prompt('Starting
CountryCode',
                  'integer',
                  '10'
)#
```

Result: select . . . where COUNTRY_MULTILINGUAL.COUNTRY_CODE > 10

promptmany

Prompt the user for one or more values. Only the prompt_name argument is required. The datatype defaults to string when not specified. The prompt is optional when defaultText is specified. The text, when specified, will precede the list of values. A queryItem can be specified to take advantage of the Prompt Info properties of the query item. The trailing_text, when specified, will be appended to the list of values.

Syntax

```
promptmany ( prompt_name , datatype , defaultText , text , queryItem ,
trailing_text )
```

Example 1

```
select . . . where COUNTRY_MULTILINGUAL.COUNTRY IN ( #promptmany ( 'CountryName'
 ) # )
```

Result: select . . . where COUNTRY_MULTILINGUAL.COUNTRY_CODE IN ('Canada' , 'The Netherlands' , 'Russia')

Example 2

```
select . . .
from
gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL,
gosales.gosales.dbo.COUNTRY XX
where COUNTRY_MULTILINGUAL.COUNTRY_CODE = XX.COUNTRY_CODE
    #promptmany('Selected CountryCodes',
                  'integer',
                  ' ',
```

```
   ' and  COUNTRY_MULTILINGUAL.COUNTRY_CODE IN (',
                  '',
                  ')'
)#
```

Result: select . . . from gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL, gosales.gosales.dbo.COUNTRY XX where COUNTRY_MULTILINGUAL.COUNTRY_CODE = XX.COUNTRY_CODE and COUNTRY_MULTILINGUAL.COUNTRY_CODE IN ('Canada' , 'The Netherlands' , 'Russia')

sb

Surround the passed string with square brackets.

Syntax

```
sb ( string_exp )
```

Example

```
#sb ( 'abc' )#
```

Result: [abc]

sq

Surround the passed string with single quotes.

Syntax

```
sq ( string_exp )
```

Example

```
#sq ( 'zero' )#
```

Result: 'zero'

sort

Sorts the elements of the array in alphabetical order. Duplicates are retained.

Syntax

```
sort ( array_exp )
```

Example

```
#csv ( sort ( array ( 's3', 'a', 'x' ) ) )#
```

Result: 'a', 's3', 'x'

split

Splits a string or the string elements of the array into separate elements.

Syntax

```
split ( pattern_string, string_exp | array_exp )
```

Example 1

```
#csv ( split ( '::', 'ab=c::de=f::gh=i' ) )#
```

Result: 'ab=c' , 'de=f', 'gh=i'

Example 2

```
#csv ( split ( '=' , split ( '::', 'ab=c::de=f::gh=i' ) ) )#
```

Result: 'ab' , 'c' , 'de' , 'f', 'gh' , 'i'

substitute

Search for a pattern in a string or in the string elements of an array and substitute the first occurrence of the found text with other text.

Syntax

```
substitute ( pattern_string, replacement_string, string_exp | array_exp )
```

Example 1

```
#sq ( substitute ( '^cn=', '***', 'cn=help' ) )#
```

Result: '***help'

Example 2

```
#csv ( substitute ( '^cn=', '***', array ( 'cn=help' , 'acn=5' ) ) )#
```

Result: '***help' , 'acn=5'

Example 3

```
#csv ( substitute ( 'cn=', '', array ( 'cn=help' , 'acn=5' ) ) )#
```

Result: 'help' , 'a5'

unique

Removes duplicate entries from the array. The order of the elements is retained.

Syntax

```
unique ( array_exp )
```

Example

```
Example:
#csv ( unique ( array ( 's3', 'a', 's3', 'x' ) ) )#
```

Result: 's3', 'a', 'x'

urlencode

URL encodes the passed argument. Useful when specifying XML connection strings.

Syntax

```
field_one=urlencode(prompt('userValue'))
```

Example

```
urlencode(prompt('some_val'))
```

Result: `%27testValue%27`

CSVIdentityName

Use the identity information of the current authenticated user to lookup values in the specified parameter map. Each individual piece of the user's identity (account name, group names, role names) is used as a key into the map. The unique list of values that is retrieved from the map is then returned as a string, where each value is surrounded by single quotes and where multiple values are separated by commas.

Syntax

```
CSVIdentityName ( %parameter_map_name [ , separator_string ] )
```

Example

```
#CSVIdentityName ( %security_clearance_level_map )#
```

Result: 'level_500' , 'level_501' , 'level_700'

CSVIdentityNameList

Returns the pieces of the user's identity (account name, group names, role names) as a list of strings. The unique list of values is returned as a string, where each value is surrounded by single quotes and where multiple values are separated by commas.

Syntax

```
CSVIdentityNameList ( [ separator_string ] )
```

Example

```
#CSVIdentityNameList ( )#
```

Result: 'Everyone' , 'Report Administrators' , 'Query User'

CAMPassport

Returns the passport.

Syntax

```
CAMPassport ( )
```

Example

```
#CAMPassport ( )#
```

Result: 111:98812d62-4fd4-037b-4354-26414cf7ebef:3677162321

CAMIDList

Returns the pieces of the user's identity (account name, group names, role names) as a list of values separated by commas.

Syntax

```
CAMIDList ( [ separator_string ] )
```

Example

```
#CAMIDList ( )#
```

Result: CAMID("::Everyone"), CAMID(":Authors"), CAMID(":Query Users"), CAMID(":Consumers"), CAMID(":Metrics Authors")

CAMIDListForType

Returns an array of the user's identities based on the identity type (account, group, or role). It can be used with the macro functions csv or join.

Syntax

```
CAMIDListForType ( identity type )
```

Example

```
[qs].[userRole] IN ( #csv ( CAMIDListForType ( 'role' ) ) # )
```

Result: [qs].[userRole] IN ('CAMID("::System Administrators")', 'CAMID(":Authors")')

Common Functions

nullif

Returns NULL if exp1 equals exp2, otherwise returns exp1.

Syntax

```
nullif( exp1, exp2 )
```

_format

Associates a format with the expression. The keyword can be PERCENTAGE_0, PERCENTAGE_1 or PERCENTAGE_2.

Syntax

```
_format ( expr , keyword  )
```

Example

```
_format( [Unit Sale Price] / [Unit Price] , PERCENTAGE_2 )
```

Result: 0.75123 displayed as 75.12%

_round

Returns the numeric expression rounded to the integer_exp places right of the decimal point. Note: integer_exp MUST be a non-negative integer.

Syntax

```
_round ( numeric_exp, integer_exp )
```

Example

```
_round ( 1220.42369, 2 )
```

Result: 1220.42

abs

Returns the absolute value of numeric_exp. The sign of negative values is changed to positive.

Syntax

```
abs ( numeric_exp )
```

Example 1

```
abs ( 15 )
```

Result: 15

Example 2

```
abs ( -15 )
```

Result: 15

ancestor

Returns the ancestor of the specified member at either the specified (named) level or the specified number of levels above the member. Note: The result is not guaranteed to be consistent when there is more than one such ancestor.

Syntax

```
ancestor ( member, level | integer )
```

ancestors

Returns all the ancestors of a member at a specified level, or distance above the member. (Most data sources support only one ancestor at a specified level, but some support more than one. Hence the result is a member set.)

Syntax

```
ancestors ( member , level | index )
```

bottomCount

This function sorts a set according to the value of "numeric_exp" evaluated at each of the members of "set_exp", and returns the bottom "index_exp" members.

Syntax

```
bottomCount ( set_exp , index_exp , numeric_exp )
```

bottomPercent

This function is similar to bottomSum, but the threshold is "numeric_exp1" percent of the total.

Syntax

```
bottomPercent ( set_exp , numeric_exp1 , numeric_exp2 )
```

bottomSum

This function sorts on "numeric_exp2", evaluated at the corresponding member of "set_exp", and picks up the bottommost elements whose cumulative total is equal to or less than numeric_exp1.

Syntax

```
bottomSum ( set_exp , numeric_exp1 , numeric_exp2 )
```

caption

Returns the caption values of the specified argument.

Syntax

```
caption ( level | member | set_exp )
```

cast

Converts an expression to a specified data type. Some data types allow for a length and precision to be specified. Make sure that the target is of the appropriate type and size. The following datatypes can be specified: CHARACTER, VARCHAR, CHAR, NUMERIC, DECIMAL, INTEGER, SMALLINT, REAL, FLOAT, DATE, TIME, TIMESTAMP, TIME WITH TIME ZONE, TIMESTAMP WITH TIME ZONE, and INTERVAL. When type casting to an INTERVAL type, one of the following interval qualifiers must be specified: YEAR, MONTH, or YEAR TO MONTH for the year-to-month interval datatype; DAY, HOUR, MINUTE, SECOND, DAY TO HOUR, DAY TO MINUTE, DAY TO SECOND, HOUR TO MINUTE, HOUR TO SECOND, or MINUTE TO SECOND for the day-to-second interval datatype. More notes for this function are included in the user guide.

Syntax

```
cast ( expression, datatype_specification )
```

Example 1

```
cast ( '123' , integer )
```

Result: 123

Example 2

```
cast ( 12345 , VARCHAR ( 10 ) )
```

Result: a string containing 12345

Notes

- When you convert a value of type TIMESTAMP to type DATE, the time portion of the timestamp value is ignored.
- When you convert a value of type TIMESTAMP to type TIME, the date portion of the timestamp is ignored.
- When you convert a value of type DATE to type TIMESTAMP, the time components of the timestamp are set to zero.
- When you convert a value of type TIME to type TIMESTAMP, the date component is set to the current system date.

- It is invalid to convert one interval datatype to the other (for instance because the number of days in a month is variable). Note that you can specify the number of digits for the leading qualifier only, i.e. YEAR(4) TO MONTH, DAY(5).
- Errors will be reported if the target type and size are not compatible with the source type and size.

ceil

Returns the smallest integer greater than or equal to numeric_exp.

Syntax

```
ceil ( numeric_exp )
```

ceiling

Returns the smallest integer greater than or equal to numeric_exp.

Syntax

```
ceiling ( numeric_exp )
```

Example 1

```
ceiling ( 4.22 )
```

Result: 5

Example 2

```
ceiling ( -1.23 )
```

Result: -1

char_length

Returns the number of characters in string_exp.

Syntax

```
char_length ( string_exp )
```

Example

```
char_length ( 'Canada' )
```

Result: 6

character_length

Returns the number of characters in string_exp.

Syntax

```
character_length ( string_exp )
```

Example

```
character_length ( 'Canada' )
```

Result: 6

children

Returns the set of children of a specified member.

Syntax

```
children ( member )
```

closingPeriod

Returns the last sibling among the descendants of a member at a specified level. Typically used with a time dimension.

Syntax

```
closingPeriod ( level [, member ] )
```

coalesce

Returns the first non-null argument (or null if all arguments are null). The Coalesce function takes two or more arguments.

Syntax

```
coalesce ( exp_list )
```

completeTuple

Similar to "tuple", identifies a cell location (intersection) based on the specified members, each of which must be from a different dimension. However, completeTuple implicitly includes the default member from all dimensions not otherwise specified in the arguments, rather than the current member. The value of this cell can be obtained with the "value" function.

Syntax

```
completeTuple ( member { , member } )
```

cousin

Returns the child member of member2 with the same relative position as the member1 is under its parent.

Syntax

```
cousin ( member1 , member2 )
```

current_date

Returns a date value representing the current date of the computer that the database software runs on.

Syntax

```
current_date
```

Example

```
current_date
```

Result: 2003-03-04

current_time

Returns a time with time zone value, representing the current time of the computer that runs the database software.

Syntax

```
current_time
```

Example

```
current_time
```

Result: 16:33:11+05:00

current_timestamp

Returns a datetime with time zone value, representing the current timestamp of the computer that runs the database software.

Syntax

```
current_timestamp
```

Example

```
current_timestamp
```

Result: 2003-03-03 16:40:15.535000+05:00

currentMember

Returns the current member of the hierarchy during an iteration. If the specified hierarchy is not present in the context in which the expression is being evaluated, its default member is assumed.

Syntax

```
currentMember ( hierarchy )
```

defaultMember

Returns the default member of a hierarchy.

Syntax

```
defaultMember ( hierarchy )
```

descendants

Returns the set of descendants of a member, or set of members, at a specified level (qualified name) or distance (integer 0..n) from the root. Multiple options may be specified (separated by a space) to determine which members are to be returned.

Syntax

```
descendants ( member | set_expr , level | distance [ , { self | before |
beforewithmember | after } ] )
```

Example

```
descendants([national].[Line].[Line].[Line1]->:[PC].[Line (Root)].[Dishwashers]
, 2, SELF AFTER)
```

Result: `Result: AcmeWash MR AcmeWash AcmeWash HE`

Notes

- self: Only the members at the specified level are included in the final set (this is the default behaviour in the absence of any options).
- before: If there are any intermediate levels between the member's level and the one specified, members from those levels are included. If the level specified is the same as the member upon which the function is applied, the member is included in the final set.
- beforewithmember: If there are any intermediate levels between the member's level and the one specified, members from those levels are included. The member upon which the function is applied is also included in the final set.
- after: If other levels exist after the specified level, members from those levels are included in the final set.

emptySet

Returns an empty member set for the specified hierarchy.

Syntax

```
emptySet ( hierarchy )
```

except

Returns the members of "set_exp1" that are not also in "set_exp2". Duplicates are retained only if the optional keyword ALL is supplied as the third argument.

Syntax

```
except ( set_exp1 , set_exp2 [,ALL] )
```

exp

Returns e raised to the power of numeric_exp. The constant e is the base of the natural logarithm.

Syntax

```
exp ( numeric_exp )
```

Example

```
exp ( 2 )
```

Result: 7.389056

extract

Returns an integer representing the value of datepart (year, month, day, hour, minute, second) in datetime_exp.

Syntax

```
extract ( datepart , datetime_exp )
```

Example 1

```
extract ( year , 2003-03-03 16:40:15.535 )
```

Result: 2003

Example 2

```
extract ( hour , 2003-03-03 16:40:15.535 )
```

Result: 16

filter

Returns the set resulting from filtering a specified set based on the boolean condition. Each member is included in the result if and only if the corresponding value of "boolean_exp" is true.

Syntax

```
filter ( set_exp , boolean_exp )
```

firstChild

Returns the first child of a member.

Syntax

```
firstChild ( member)
```

firstSibling

Returns the first child of the parent of a member.

Syntax

```
firstSibling ( member )
```

floor

Returns the largest integer less than or equal to numeric_exp.

Syntax

```
floor ( numeric_exp )
```

Example 1

```
floor ( 3.22 )
```

Result: 3

Example 2

```
floor ( -1.23 )
```

Result: -2

generate

This function evaluates "set_exp2" for each member of "set_exp1" and joins the resulting sets by union. If ALL is specified, duplicates in the result are retained.

Syntax

```
generate ( set_exp1 , set_exp2 [ , ALL ] )
```

head

Returns the first "index_exp" elements of "set_exp". The default for "index_exp" is 1.

Syntax

```
head ( set_exp [ , index_exp ] )
```

hierarchize

This function orders the members of a set in a hierarchy. Members in a level are sorted in their natural order, which is the default ordering of the members along a dimension when no other sort conditions are specified.

Syntax

```
hierarchize ( set_exp )
```

hierarchy

Returns the hierarchy that contains the specified level, member or member set.

Syntax

```
hierarchy ( level | member | set_exp )
```

intersect

Returns the intersection of two input sets. The result retains duplicates only when the optional keyword ALL is supplied as the third argument.

Syntax

```
intersect ( set_exp1 , set_exp2 [ , ALL ] )
```

item

Returns a member from a specified location within a set. The index into the set is zero based.

Syntax

```
item ( set_exp , index )
```

lag

Returns the sibling member that is a specified number of positions prior to a specified member.

Syntax

```
lag ( member , index_exp )
```

lastChild

Returns the last child of a specified member.

Syntax

```
lastChild ( member )
```

lastPeriods

Returns the set of members from the same level that ends with the specified member. The number of members returned is the absolute value of "integer_exp". If "integer_exp" is negative, members following and including the specified member are returned. Typically used with a time dimension.

Syntax

```
lastPeriods ( integer_exp , member )
```

lastSibling

Returns the last child of the parent of a specified member.

Syntax

```
lastSibling ( member )
```

lead

Returns the sibling member that is a specified number of positions following a specified member.

Syntax

```
lead ( member , index_exp )
```

level

Returns the level of a member.

Syntax

```
level ( member )
```

levels

Returns the level in the hierarchy whose distance from the root is specified by "index".

Syntax

```
levels ( hierarchy , index )
```

linkMember

Returns the corresponding member in the specified level or hierarchy (of the same dimension). For level-based hierarchies, a level must be specified as the second argument and for parent-child hierarchies, a hierarchy must be specified. An exception is thrown when the second parameter does not resolve to a hierarchy of the dimension that the member of the first parameter belongs to.

Syntax

```
linkMember ( member , level | hierarchy )
```

ln

Returns the natural logarithm of the numeric_exp.

Syntax

```
ln ( numeric_exp )
```

Example

```
ln ( 4 )
```

Result: 1.38629

localtime

Returns a time value, representing the current time of the computer that runs the database software.

Syntax

```
localtime
```

Example

```
localtime
```

Result: 16:33:11

localtimestamp

Returns a datetime value, representing the current timestamp of the computer that runs the database software.

Syntax

```
localtimestamp
```

Example

```
localtimestamp
```

Result: 2003-03-03 16:40:15.535000

lower

Returns string_exp with all uppercase characters shifted to lowercase.

Syntax

```
lower ( string_exp )
```

Example

```
lower ( 'ABCDEF' )
```

Result: 'abcdef'

member

Defines a member based on the specified expression in the specified hierarchy. "string1" is used to identify the member created by this function it must be unique in the query, and must be different from any other member in the same hierarchy. "string2" is used as the caption of the member; if it is absent, the caption is empty. If the hierarchy is omitted, the measure dimension is assumed. Note: All calculations used as grouping items whose sibling items are other calculations or member sets, should be explicitly assigned to a hierarchy using this function, otherwise the results are not predictable. The only exception to this is where the calculation involves only members of the same hierarchy as the siblings. In that case the calculation is assumed to belong to that hierarchy.

Syntax

```
member ( value_exp [ , string1 [ , string2 [ , hierarchy ] ] ] )
```

members

Returns the set of members in a hierarchy or level. In the case of a hierarchy, the order of the members in the result is not guaranteed; if a predictable order is required, an explicit ordering function (such as hierarchize) must be used.

Syntax

```
members ( hierarchy | level )
```

mod

Returns the remainder (modulus) of integer_exp1 divided by integer_exp2. The integer_exp2 must not be zero or an exception condition is raised.

Syntax

```
mod ( integer_exp1, integer_exp2 )
```

Example

```
mod ( 20 , 3 )
```

Result: 2

nestedSet

Returns the set of members of set_expr2 evaluated in the context of the current member of set_expr1.

Syntax

```
nestedSet ( set_expr1 , set_expr2 )
```

nextMember

Returns the next member in the level to which the specified member exists.

Syntax

```
nextMember ( member )
```

octet_length

Returns the number of bytes in string_exp.

Syntax

```
octet_length ( string_exp )
```

Example 1

```
octet_length ( 'ABCDEF' )
```

Result: 6

Example 2

```
octet_length ( '' )
```

Result: 0

openingPeriod

Returns the first sibling member among the descendants of a member at a specified level. Typically used with a time dimension.

Syntax

```
openingPeriod ( level [ , member ] )
```

order

Arranges members of a specified set, as determined from the set of values created by evaluating "value_exp" for each value of the set, and modified by the third parameter. There are two varieties of order: hierarchized (ASC or DESC) and non-hierarchized (BASC or BDESC, where B stands for "break hierarchy"). The hierarchized ordering first arranges members according to their position in the hierarchy. Then it orders the children of each member according to "value_exp". The non-hierarchized ordering arranges members in the set without regard to the hierarchy. In the absence of an explicit specification, ASC is the default.

Syntax

```
order ( set_exp , value_exp [ , ASC | DESC | BASC | BDESC ] )
```

ordinal

Returns the zero-based ordinal value (distance from the root level) of the specified level.

Syntax

```
ordinal ( level )
```

parallelPeriod

Returns a member from a prior period in the same relative position as a specified member. This function is similar to the "Cousin" function, but is more closely related to time series. It takes the ancestor of "member" at "level" (call it "ancestor"); then it takes the sibling of "ancestor" that lags by "int_exp" positions, and returns the parallel period of "member" among the descendants of that sibling. When unspecified, "int_exp" defaults to 1 and "member" defaults to the current member.

Syntax

```
parallelPeriod ( level [ , int_exp [ , member ] ] )
```

parent

Returns the member that is the parent of the specified member.

Syntax

```
parent ( member )
```

periodsToDate

Returns a set of sibling members from the same level as a given member, as constrained by a specified level. It locates the ancestor of "member" at "level", and returns that ancestor's descendants at the same level as "member", up to and including "member". Typically used with a time dimension.

Syntax

```
periodsToDate ( level , member )
```

position

Returns the integer value representing the starting position of the first string_exp in the second string_exp or 0 when the first string_exp is not found.

Syntax

```
position ( string_exp , string_exp )
```

Example 1

```
position ( 'C' , 'ABCDEF' )
```

Result: 3

Example 2

```
position ( 'H' , 'ABCDEF' )
```

Result: 0

power

Returns numeric_exp1 raised to the power numeric_exp2. If numeric_exp1 is negative then numeric_exp2 must result in an integer value.

Syntax

```
power ( numeric_exp1, numeric_exp2 )
```

Example

```
power ( 3 , 2 )
```

Result: 9

prevMember

Returns the member that immediately precedes the specified member in the same level.

Syntax

```
prevMember ( member )
```

roleValue

Returns the value of the attribute that is associated with the role whose name is specified by "string" within the specified context. The second argument is optional only in a number of limited circumstances, where it can be derived from other context. Applications can be made portable across different data sources and models by accessing attributes by role, rather than by query item ID. (For dimensionally modelled relational data sources, assignment of roles is the modeller's responsibility.) Intrinsic roles that are defined for members of all data source types include: '_businessKey', '_memberCaption', '_memberDescription', '_memberUniqueName'.

Syntax

```
roleValue ( string [ , member | set_exp ] )
```

Example

```
roleValue ( '_memberCaption', [Sales].[Product].[Product].[Product line]->[all]
.[1] )
```

Result: Camping Equipment

rootMembers

Returns the root members of a hierarchy.

Syntax

```
rootMembers ( hierarchy )
```

set

Returns the list of members defined in the expression. The members must belong to the same hierarchy.

Syntax

```
set ( member { , member } )
```

siblings

Returns the children of the parent of the specified member.

Syntax

```
siblings ( member )
```

sqrt

Returns the square root of numeric_exp. numeric_exp must be non-negative.

Syntax

```
sqrt ( numeric_exp )
```

Example

```
sqrt ( 9 )
```

Result: 3

subset

Returns a subset of members from a specified set starting "index_exp1" from the beginning. If the count "index_exp2" is specified, that many members (if available) are returned. Otherwise, all remaining members are returned.

Syntax

```
subset ( set_exp, index_exp1 [ , index_exp2 ] )
```

substring

Returns the substring of string_exp that starts at position integer_exp1 for integer_exp2 characters or to the end of string_exp if integer_exp2 is omitted. The first character in string_exp is at position 1.

Syntax

```
substring ( string_exp , integer_exp1 [ , integer_exp2 ] )
```

Example

```
substring ( 'abdefg', 3, 2)
```

Result: 'de'

tail

Returns the last "index_exp" elements of "set exp". The default for "index_exp" is 1.

Syntax

```
tail ( set_exp [ , index_exp ] )
```

topCount

This function sorts a set according to the values of "numeric_exp" evaluated at each of the members of "set_exp", and returns the top "index_exp" members.

Syntax

```
topCount ( set_exp , index_exp , numeric_exp )
```

topPercent

This function is similar to topSum, but the threshold is "numeric_exp1" percent of the total.

Syntax

```
topPercent ( set_exp , numeric_exp1, numeric_exp2 )
```

topSum

This function sorts on "numeric_exp2", evaluated at the corresponding members of "set_exp", and picks up the topmost elements whose cumulative total is at least "numeric_exp1".

Syntax

```
topSum ( set_exp , numeric_exp1 , numeric_exp2 )
```

trim

Returns a string_exp trimmed of leading and\or trailing blanks or trimmed of a certain character specified in match_character_exp. BOTH is implicit when first argument is not stated and blank is implicit when second argument is not stated.

Syntax

```
trim ( [ [ TRAILING | LEADING | BOTH ] [ match_character_exp ] , ] string_exp
)
```

Example 1

```
trim ( TRAILING 'A' , 'ABCDEFA' )
```

Result: 'ABCDEF'

Example 2

```
trim ( BOTH ' ABCDEF ' )
```

Result: 'ABCDEF'

tuple

Identifies a cell location (intersection) based on the specified members, each of which must be from a different dimension. Implicitly includes the current member from all dimensions not otherwise specified in the arguments. The current member of any dimension not specified in the evaluating context is assumed to be the default member of that dimension. The value of this cell can be obtained with the "value" function.

Syntax

```
tuple (  member { , member } )
```

union

This function returns the union of 2 sets "set_exp1" and "set_exp2". The result retains duplicates only when the optional keyword ALL is supplied as the third argument.

Syntax

```
union ( set_exp1 , set_exp2 [ , ALL ] )
```

unique

Removes all duplicates from the specified set. The remaining members retain their original order.

Syntax

```
unique ( set_expr )
```

upper

Returns string_exp with all lowercase characters shifted to uppercase.

Syntax

```
upper ( string_exp )
```

Example

```
upper ( 'abcdef' )
```

Result: 'ABCDEF'

value

Returns the value of the cell identified by a tuple. Note that the default member of the Measures dimension is the Default Measure.

Syntax

```
value ( tuple )
```

DB2

ascii

Returns the ASCII code value of the leftmost character of the argument as an integer.

Syntax

```
ascii ( string_exp )
```

ceiling

Returns the smallest integer greater than or equal to numeric_exp.

Syntax

```
ceiling ( numeric_exp )
```

char

Returns a string representation of a date/time value or a decimal number.

Syntax

```
char ( exp )
```

chr

Returns the character that has the ASCII code value specified by integer_exp. integer_exp should be between 0 and 255.

Syntax

```
chr ( integer_exp )
```

concat

Returns a string that is the result of concatenating string_exp1 with string_exp2.

Syntax

```
concat ( string_exp1, string_exp2 )
```

date

Returns a date from a single input value. exp can be a string or integer representation of a date.

Syntax

```
date ( exp )
```

day

Returns the day of the month (1-31) from date_exp. date_exp can be a date value or a string representation of a date.

Syntax

```
day ( date_exp )
```

dayname

Returns a character string containing the data source_specific name of the day (for example, Sunday through Saturday or Sun. through Sat. for a data source that uses English, or Sonntag through Samstag for a data source that uses German) for the day portion of date_exp. date_exp can be a date value or a string representation of a date.

Syntax

```
dayname ( date_exp )
```

dayofweek

Returns the day of the week in date_exp as an integer in the range 1 to 7, where 1 represents Sunday. date_exp can be a date value or a string representation of a date.

Syntax

```
dayofweek ( date_exp )
```

dayofweek_iso

Returns the day of the week in date_exp as an integer in the range 1 to 7, where 1 represents Monday. date_exp can be a date value or a string representation of a date.

Syntax

```
dayofweek_iso ( date_exp )
```

dayofyear

Returns the day of the year in date_exp as an integer in the range 1 to 366. date_exp can be a date value or a string representation of a date.

Syntax

```
dayofyear ( date_exp )
```

days

Returns an integer representation of a date. exp can be a date value or a string representation of a date.

Syntax

```
days ( exp )
```

dec

Returns decimal representation of string_exp1 with precision numeric_exp1, scale numeric_exp2 and decimal character string_exp2. String_exp1 must be formatted as a SQL Integer or Decimal constant.

Syntax

```
dec ( string_exp1 [ , numeric_exp1 [ , numeric_exp2 [ , string_exp2 ] ] ] )
```

decimal

Returns decimal representation of string_exp1 with precision numeric_exp1, scale numeric_exp2 and decimal character string_exp2. String_exp1 must be formatted as a SQL Integer or Decimal constant.

Syntax

```
decimal ( string_exp1 [ , numeric_exp1 [ , numeric_exp2 [ , string_exp2 ] ] ]
)
```

difference

Returns an integer value representing the difference between the values returned by the data source_specific soundex function for string_exp1 and string_exp2. The value returned ranges from 0 to 4, with 4 indicating the best match. Note that 4 does not mean that the strings are equal.

Syntax

```
difference ( string_exp1, string_exp2 )
```

digits

Returns the character string representation of a non-floating point number.

Syntax

```
digits ( numeric_exp )
```

double

Returns the floating-point representation of an expression. 'exp' can be either a numeric or string expression.

Syntax

```
double ( exp )
```

event_mon_state

Returns the operational state of a particular state monitor.

Syntax

```
event_mon_state ( string_exp )
```

float

Returns the floating-point representation of a number.

Syntax

```
float ( numeric_exp )
```

hex

Returns the hexadecimal representation of a value.

Syntax

```
hex ( exp )
```

hour

Returns the hour (an integer from 0, which is midnight, to 23, which is 11:00 pm) from time_exp. time_exp can be a time value or a string representation of a time.

Syntax

```
hour ( time_exp )
```

insert

Returns a string where length (integer_exp2) characters have been deleted from string_exp1 beginning at start (integer_exp1) and where string_exp2 has been inserted into string_exp1 at start. The first character in a string is at position 1.

Syntax

```
insert ( string_exp1, integer_exp1, integer_exp2, string_exp2 )
```

integer

Returns the integer representation of an expression. exp can be a numeric value or a string representation of a number.

Syntax

```
integer ( exp )
```

int

Returns the integer representation of an expression. exp can be a numeric value or a string representation of a number.

Syntax

```
int ( exp )
```

julian_day

Returns an integer value representing the number of days from January 1, 4712 BC (the start of the Julian date calendar) to the date value specified in exp. exp can be a date value or a string representation of a date.

Syntax

```
julian_day ( exp )
```

lcase

Returns string_exp with all uppercase characters shifted to lowercase.

Syntax

```
lcase ( string_exp )
```

left

Returns the leftmost integer_exp characters of string_exp.

Syntax

```
left ( string_exp, integer_exp )
```

length

Returns the length of the operand in bytes (except for double byte string types which return the length in characters).

Syntax

```
length ( exp )
```

locate

Returns the starting position of the first occurrence of string_exp1 within string_exp2. The search starts at position start (integer_exp) of string_exp2. The first character in a string is at position 1. If string_exp1 is not found then zero is returned.

Syntax

```
locate ( string_exp1, string_exp2 [ , integer_exp ] )
```

long_varchar

Returns a long string.

Syntax

```
long_varchar ( string_exp )
```

ltrim

Returns string_exp with leading spaces removed.

Syntax

```
ltrim ( string_exp )
```

microsecond

Returns the microsecond (time-unit) part of a value. exp can be a timestamp or a string representation of a timestamp.

Syntax

```
microsecond ( exp )
```

midnight_seconds

Returns an integer value in the range 0 to 86400 representing the number of seconds between midnight and time value specified in the argument. exp can be a time value, a timestamp or a string representation of a time.

Syntax

```
midnight_seconds ( exp )
```

minute

Returns the minute (an integer from 0-59) from time_exp. time_exp can be a time value, a timestamp or a string representation of a time.

Syntax

```
minute ( time_exp )
```

month

Returns the month (an integer from 1-12) from date_exp.

Syntax

```
month ( date_exp )
```

monthname

Returns a character string containing the data source_specific name of the month (for example, January through December or Jan. through Dec. for a data source that uses English, or Januar through Dezember for a data source that uses German) for the month portion of date_exp.

Syntax

```
monthname ( date_exp )
```

quarter

Returns the quarter in date_exp as a number in the range 1 to 4, where 1 represents January 1 through March 31.

Syntax

```
quarter ( date_exp )
```

radians

Returns the number of radians converted from numeric_exp degrees.

Syntax

```
radians ( numeric_exp )
```

repeat

Returns a string consisting of string_exp repeated integer_exp times.

Syntax

```
repeat ( string_exp, integer_exp )
```

replace

Replaces all occurrences of string_exp2 in string_exp1 with string_exp3.

Syntax

```
replace ( string_exp1, string_exp2, string_exp3 )
```

right

Returns the rightmost integer_exp characters of string_exp.

Syntax

```
right ( string_exp, integer_exp )
```

round

Returns numeric_exp rounded to the nearest value integer_exp places right of the decimal point. If integer_exp is negative, numeric_exp is rounded to the nearest value absolute (integer_exp) places to the left of the decimal point, e.g., round-near (125, -1) rounds to 130.

Syntax

```
round ( numeric_exp, integer_exp )
```

rtrim

Returns string_exp with trailing spaces removed.

Syntax

```
rtrim ( string_exp )
```

second

Returns the second (an integer from 0-59) from time_exp.

Syntax

```
second ( time_exp )
```

sign

Returns an indicator of the sign of numeric_exp: +1 if numeric_exp is positive, 0 if zero or -1 if negative.

Syntax

```
sign ( numeric_exp )
```

smallint

Returns the small integer representation of a number.

Syntax

```
smallint ( exp )
```

soundex

Returns a 4 character string code obtained by systematically abbreviating words and names in string_exp according to phonetics. Can be used to determine if two strings sound the same, e.g., does sound-of ('SMITH') = sound-of ('SMYTH').

Syntax

```
soundex ( string_exp )
```

space

Returns a string consisting of integer_exp spaces.

Syntax

```
space ( integer_exp )
```

substr

Returns the substring of string_exp that starts at position integer_exp1 for integer_exp2 characters. The first character in string_exp is at position 1.

Syntax

```
substr ( string_exp, integer_exp1 [ , integer_exp2 ] )
```

table_name

Returns an unqualified name of a table or view based on the object name in string_exp1 and the schema name given in string_exp2. It is used to resolve aliases.

Syntax

```
table_name ( string_exp1 [ , string_exp2 ] )
```

table_schema

Returns the schema name portion of the two part table or view name based on the object name in string_exp1 and the schema name in string_exp2. It is used to resolve aliases.

Syntax

```
table_schema ( string_exp1 [ , string_exp2 ] )
```

time

Returns a time from a value.

Syntax

```
time ( exp )
```

timestamp

Returns a timestamp from a value or a pair of values. exp1 must represent a date value, and exp2 must represent a time value.

Syntax

```
timestamp ( exp1 [ , exp2 ] )
```

timestamp_iso

Returns a datetime in the ISO format (yyyy-mm-dd hh:mm:ss.nnnnnn) converted from the IBM format (yyyy-mm-dd-hh.mm.ss.nnnnnn). If the exp is a time, it inserts the value of the CURRENT DATE for the date elements and zero for the fractional time element.

Syntax

```
timestamp_iso ( exp )
```

timestampdiff

Returns an estimated number of intervals of type exp1 based on the difference between two timestamps. Exp2 is the result of subtracting two timestamp types and converting the result to CHAR. Valid values of exp1 are: 1 Fractions of a second; 2 Seconds; 4 Minutes; 8 Hours; 16 Days; 32 Weeks; 64 Months; 128 Quarters; 256 Years.

Syntax

```
timestampdiff ( exp1, exp2 )
```

to_char

Returns the string representation of a timestamp with the format of string_exp.

Syntax

```
to_char ( timestamp_exp , string_exp )
```

translate

Returns string_exp1 in which characters from string_exp3 are translated to the equivalent characters in string_exp2. string_exp4 is a single character that is used to pad string_exp2 if it is shorter than string_exp3. If only string_exp1 is present, then this function translates it to uppercase characters.

Syntax

```
translate ( string_exp1 [ , string_exp2, string_exp3 [ , string_exp4 ] ] )
```

trunc

Returns numeric_exp1 truncated to numeric_exp2 places RIGHT of the decimal point. If numeric_exp2 is negative, numeric_exp1 is truncated to the absolute value of numeric_exp2 places to the LEFT of the decimal point.

Syntax

```
trunc ( numeric_exp1, numeric_exp2 )
```

truncate

Returns numeric_exp1 truncated to numeric_exp2 places RIGHT of the decimal point. If numeric_exp2 is negative, numeric_exp1 is truncated to the absolute value of numeric_exp2 places to the LEFT of the decimal point.

Syntax

```
truncate ( numeric_exp1, numeric_exp2 )
```

ucase

Returns string_exp with all lowercase characters shifted to uppercase.

Syntax

```
ucase ( string_exp )
```

value

Returns the first non-null argument (or null if all arguments are null). The Value function takes two or more arguments.

Syntax

```
value ( exp_list )
```

varchar

Returns a VARCHAR representation of exp, with length numeric_exp.

Syntax

```
varchar ( exp [ , numeric_exp ] )
```

week

Returns the week of the year in date_exp as an integer value in the range 1 to 53.

Syntax

```
week ( date_exp )
```

year

Returns the year from date_exp.

Syntax

```
year ( date_exp )
```

DB2 Cast

cast_char

Returns the first numeric_exp characters of the value of exp cast as a string. The whole string is returned when the second argument is not specified.

Syntax

```
cast_char ( exp [ , numeric_exp ] )
```

cast_date

Returns the value of the expression cast as a date.

Syntax

```
cast_date ( exp )
```

cast_decimal

Returns the value of exp cast as a decimal with the precision of numeric_exp1 and scale of numeric_exp2.

Syntax

```
cast_decimal ( exp [ , numeric_exp1, numeric_exp2 ] )
```

cast_double_precision

Returns the value of the expression cast as a double.

Syntax

```
cast_double_precision ( exp )
```

cast_float

Returns the value of the expression cast as a float.

Syntax

```
cast_float ( exp )
```

cast_integer

Returns the value of the expression cast as a integer.

Syntax

```
cast_integer ( exp )
```

cast_longvarchar

Returns the value of the expression cast as a longvarchar.

Syntax

```
cast_longvarchar ( string_exp )
```

cast_smallint

Returns the value of the expression cast as a smallint.

Syntax

```
cast_smallint ( exp )
```

cast_time

Returns the value of the expression cast as a time value.

Syntax

```
cast_time ( string_exp )
```

cast_timestamp

Returns the value of the expression cast as a datetime.

Syntax

```
cast_timestamp ( exp )
```

cast_varchar

Returns the value of the expression cast as a varchar with length.

Syntax

```
cast_varchar ( exp, integer_exp )
```

DB2 Math

log

Returns the natural logarithm of numeric_exp.

Syntax

```
log ( numeric_exp )
```

log10

Returns the base ten logarithm of numeric_exp.

Syntax

```
log10 ( numeric_exp )
```

rand

Generates a random number using integer_exp as a seed value.

Syntax

```
rand ( integer_exp )
```

DB2 Trigonometry

acos

Returns the arccosine of numeric_exp in radians. The arccosine is the angle whose cosine is numeric_exp.

Syntax

```
acos ( numeric_exp )
```

asin

Returns the arcsine of numeric_exp in radians. The arcsine is the angle whose sine is numeric_exp.

Syntax

```
asin ( numeric_exp )
```

atan

Returns the arctangent of numeric_exp in radians. The arctangent is the angle whose tangent is numeric_exp.

Syntax

```
atan ( numeric_exp )
```

atanh

Returns the hyperbolic arctangent of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
atanh (numeric_exp )
```

atan2

Returns the arctangent of the x and y coordinates specified by numeric_exp1 and numeric_exp2, respectively, in radians. The arctangent is the angle whose tangent is numeric_exp2 / numeric_exp1.

Syntax

```
atan2 ( numeric_exp1, numeric_exp2 )
```

cos

Returns the cosine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
cos ( numeric_exp )
```

cosh

Returns the hyperbolic cosine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
cosh ( numeric_exp )
```

cot

Returns the cotangent of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
cot ( numeric_exp )
```

degrees

Returns numeric_exp radians converted to degrees.

Syntax

```
degrees ( numeric_exp )
```

sin

Returns the sine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
sin ( numeric_exp )
```

sinh

Returns the hyperbolic sine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
sinh ( numeric_exp )
```

tan

Returns the tangent of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
tan ( numeric_exp )
```

tanh

Returns the hyperbolic tangent of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
tanh ( numeric_exp )
```

Informix

cardinality

Returns the number of elements in a collection column (SET, MULTISET, LIST).

Syntax

```
cardinality ( string_exp )
```

char_length

Returns the number of logical characters (which can be distinct from the number of bytes in some East Asian locales) in string_exp.

Syntax

```
char_length ( string_exp )
```

concat

Returns a string that is the result of concatenating string_exp1 to string_exp2.

Syntax

```
concat ( string_exp1, string_exp2 )
```

date

Returns the date value of either string_exp or date_exp or int_exp.

Syntax

```
date ( string_exp | date_exp | int_exp )
```

day

Returns an integer that represents the day of the month.

Syntax

```
day ( date_exp )
```

extend

The extend function adjusts the precision of a DATETIME or DATE expression. The expression cannot be a quoted string representation of a DATE value. If you do not specify first and last qualifiers, the default qualifiers are YEAR TO FRACTION(3). If the expression contains fields that are not specified by the qualifiers, the unwanted fields are discarded. If the first qualifier specifies a larger (that is, more significant) field than what exists in the expression, the new fields are filled in with values returned by the CURRENT function. If the last qualifier specifies a smaller field (that is, less significant) than what exists in the expression, the new fields are filled in with constant values. A missing MONTH or DAY field is filled in with 1, and the missing HOUR to FRACTION fields are filled in with 0.

Syntax

```
extend ( date_exp , ' { ' YEAR TO SECOND ' } ' )
```

Example

```
EXTEND (some_date_column , {YEAR TO SECOND} )
```

hex

Returns the hexadecimal encoding of an integer integer_exp.

Syntax

```
hex ( integer_exp )
```

initcap

Returns string_exp, with the first letter of each word in uppercase, all other letters in lowercase. With this function, a word begins after any character other than a letter. Thus, in addition to a blank space, symbols such as commas, periods, colons, and so on, introduce a new word.

Syntax

```
initcap ( string_exp )
```

length

Returns the number of bytes in string_exp, which is not including any trailing blank spaces. For BYTE or TEXT string_exp, LENGTH returns the full number of bytes, including any trailing blank spaces.

Syntax

```
length ( string_exp )
```

lpad

Returns a copy of string_exp1 that is left-padded (string_exp2) to the total number of characters specified by integer_exp. The sequence of string_exp2 occurs as many times as necessary to make the return string the length specified by integer_exp.

Syntax

```
lpad ( string_exp1, integer_exp, string_exp2 )
```

mdy

Returns a type DATE value with three expressions that evaluate to integers that represent the month (integer_exp1), day(integer_exp2), and year(integer_exp3).

Syntax

```
mdy ( integer_exp1, integer_exp2, integer_exp3 )
```

month

Returns an integer corresponding to the month portion of date_exp.

Syntax

```
month ( date_exp )
```

nvl

Returns the value of exp1 if exp1 is not NULL. If exp1 is NULL, then the value of exp2 is returned.

Syntax

```
nvl ( exp1, exp2 )
```

octet_length

Returns the number of bytes in string_exp, including any trailing spaces.

Syntax

```
octet_length ( string_exp )
```

replace

Returns a copy of string_exp1 in which every occurrence of string_exp2 is replaced by string_exp3. If you omit the string_exp3 option, every occurrence of string_exp2 is omitted from the return string.

Syntax

```
replace ( string_exp1, string_exp2 [ , string_exp3 ] )
```

round

Returns the rounded value of an numeric_exp. If you omit the integer_exp, the value is rounded to zero digits or to the units place. The digit range of 32 (+ and -) refers to the entire decimal value.

Syntax

```
round ( numeric_exp [ , integer_exp ] )
```

rpad

Returns a copy of string_exp1 that is right-padded (string_exp2) to the total number of characters specified by integer_exp. The sequence of string_exp2 occurs as many times as necessary to make the return string the length specified by integer_exp.

Syntax

```
rpad ( string_exp1, integer_exp, string_exp2 )
```

substr

Returns the substring of string_exp that starts at position integer_exp1. The first character in string_exp is at position 1. integer_exp2 can be used to select fewer characters, by default it selects character to the end of the string.

Syntax

```
substr ( string_exp, integer_exp1 [ , integer_exp2 ] )
```

to_char

Returns the character string of date_exp with the specified string_exp formatting. You can use this function only with built-in data types.

Syntax

```
to_char ( date_exp, string_exp )
```

to_date

Returns the string_exp1 as a date according to the date format you specify in the string_exp2 parameter. If string_exp1 is NULL, then a NULL value is returned.

Syntax

```
to_date ( string_exp1, string_exp2 )
```

trunc

Returns the truncated value of a numeric_exp. If you omit integer_exp, the value is truncated to zero digits or to the unit's place. The digit limitation of 32 (+ and -) refers to the entire decimal value.

Syntax

```
trunc ( numeric_exp [ , integer_exp ] )
```

weekday

Returns an integer that represents the day of the week; zero (0) represents Sunday, one (1) represents Monday, and so on.

Syntax

```
weekday ( date_exp )
```

year

Returns a four-digit integer that represents the year.

Syntax

```
year ( date_exp )
```

Informix Math

log10

Returns the log of a numeric_exp to base 10.

Syntax

```
log10 ( numeric_exp )
```

logn

Returns the natural logarithm of a numeric_exp.

Syntax

```
logn ( numeric_exp )
```

root

Returns the root value of a numeric_exp. Requires at least one numeric argument (the radians argument). If only the numeric_exp1 is supplied, the value 2 is used as a default value for numeric_exp2; 0 cannot be used as the value of numeric_exp2.

Syntax

```
root ( numeric_exp1[ , numeric_exp2 ] )
```

Informix Trigonometry

acos

Returns the arccosine of numeric_exp in radians. The arccosine is the angle whose cosine is numeric_exp.

Syntax

```
acos ( numeric_exp )
```

asin

Returns the arcsine of numeric_exp in radians. The arcsine is the angle whose sine is numeric_exp.

Syntax

```
asin ( numeric_exp )
```

atan

Returns the arctangent of numeric_exp in radians. The arctangent is the angle whose tangent is numeric_exp.

Syntax

```
atan ( numeric_exp )
```

atan2

Returns the arctangent of the x and y coordinates specified by numeric_exp1 and numeric_exp2, respectively, in radians. The arctangent is the angle whose tangent is numeric_exp1.

Syntax

```
atan2 ( numeric_exp1, numeric_exp2 )
```

cos

Returns the cosine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
cos ( numeric_exp )
```

sin

Returns the sine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
sin ( numeric_exp )
```

tan

Returns the tangent of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
tan ( numeric_exp )
```

MS Access

ascii

Returns a number representing the ascii code value of the leftmost character of string_exp.

Syntax

```
ascii(string_exp)
```

ceiling

Returns the smallest integer greater than or equal to numeric_exp.

Syntax

```
ceiling(numeric_exp)
```

chr

Returns the character that has the ASCII code value specified by integer_exp. integer_exp should be between 0 and 255.

Syntax

```
chr(integer_exp)
```

concat

Returns a string that is the result of concatenating string_exp1 to string_exp2.

Syntax

```
concat(string_exp1, string_exp2)
```

curdate

Returns a date value representing the current date of the computer that the database software runs on.

Syntax

```
curdate()
```

curtime

Returns a time value representing the current time of the computer that the database software runs on.

Syntax

```
curtime()
```

dayname

Returns a character string containing the data source_specific name of the day (for example, Sunday through Saturday or Sun. through Sat. for a data source that uses English, or Sonntag through Samstag for a data source that uses German) for the day portion of date_exp.

Syntax

```
dayname(date_exp)
```

dayofmonth

Returns the day of the month (1-31) from date_exp. Returns the days field (a signed integer) from interval_exp.

Syntax

```
dayofmonth(date_exp|interval_exp)
```

dayofweek

Returns the day of the week in date_exp as an integer in the range 1 to 7, where 1 represents Monday.

Syntax

```
dayofweek(date_exp)
```

dayofyear

Returns the day of the year in date_exp as an integer in the range 1 to 366.

Syntax

```
dayofyear(date_exp)
```

hour

Returns the hour (an integer from 0, which is midnight, to 23, which is 11:00 pm) from time_exp.

Syntax

```
hour(time_exp)
```

instr

Searches string_exp1 for the first occurrence of string_exp2. Returns an integer specifying the position of string_exp2. The optional argument integer_exp1 sets the starting position for the search. If omitted, the search begins at the first character position of string_exp1. The optional argument integer_exp2 specifies the type of string comparison. integer_exp1 is required if integer_exp2 is specified.

Syntax

```
instr ( [ integer_exp1 , ] string_exp1, string_exp2 [ , integer_exp2 ] )
```

lcase

Returns string_exp with all uppercase characters shifted to lowercase.

Syntax

```
lcase(string_exp)
```

left

Returns the leftmost integer_exp characters of string_exp.

Syntax

```
left(string_exp, integer_exp)
```

length

Returns the number of characters in string_exp, excluding trailing blanks and the string termination character.

Syntax

```
length(string_exp)
```

locate

Returns the starting position of the first occurrence of string_exp1 within string_exp2. The search starts at position start (integer_exp) of string_exp2. The first character in a string is at position 1. If string_exp1 is not found then zero is returned.

Syntax

```
locate(string_exp1, string_exp2 [ , integer_exp ] )
```

ltrim

Returns string_exp with leading spaces removed.

Syntax

```
ltrim(string_exp)
```

minute

Returns the minute (an integer from 0-59) from time_exp.

Syntax

```
minute(time_exp)
```

month

Returns the month (an integer from 1-12) from date_exp.

Syntax

```
month(date_exp)
```

monthname

Returns a character string containing the data source_specific name of the month (for example, January through December or Jan. through Dec. for a data source that uses English, or Januar through Dezember for a data source that uses German) for the month portion of date_exp.

Syntax

```
monthname(date_exp)
```

now

Returns a datetime value representing the current date and time of the computer that the database software runs on.

Syntax

```
now()
```

position

Returns the starting position of string_exp1 in string_exp2. The first character in a string is at position 1.

Syntax

```
position(string_exp1, string_exp2)
```

quarter

Returns the quarter in date_exp as a number in the range 1 to 4, where 1 represents January 1 through March 31.

Syntax

```
quarter(date_exp)
```

right

Returns the rightmost integer_exp characters of string_exp.

Syntax

```
right(string_exp, integer_exp)
```

round

Returns numeric_exp rounded to the nearest value integer_exp places right of the decimal point. If integer_exp is negative, numeric_exp is rounded to the nearest value absolute (integer_exp) places to the left of the decimal point.

Syntax

```
round(numeric_exp, integer_exp)
```

rtrim

Returns string_exp with trailing spaces removed.

Syntax

```
rtrim(string_exp)
```

sign

Returns an indicator of the sign of numeric_exp: +1 if numeric_exp is positive, 0 if zero or -1 if negative.

Syntax

```
sign(numeric_exp)
```

space

Returns a string consisting of integer_exp spaces.

Syntax

```
space(integer_exp)
```

substr

Returns the substring of string_exp that starts at position integer_exp1 for integer_exp2 characters. The first character in string_exp is at position 1.

Syntax

```
substr(string_exp, integer_exp1, integer_exp2)
```

substring

Returns the substring of string_exp that starts at position integer_exp1 for integer_exp2 characters. The first character in string_exp is at position 1.

Syntax

```
substring(string_exp, integer_exp1, integer_exp2)
```

truncate

Returns string_exp with trailing spaces removed.

Syntax

```
truncate(string_exp)
```

ucase

Returns string_exp with all lowercase characters shifted to uppercase.

Syntax

```
ucase(string_exp)
```

week

Returns the week of the year in date_exp as an integer value in the range 1 to 53.

Syntax

```
week(date_exp)
```

year

Returns the year from date_exp.

Syntax

```
year(date_exp)
```

MS Access Cast

cast_decimal

Returns the value of the expression cast as a decimal.

Syntax

```
cast_decimal(exp)
```

cast_float

Returns the value of the expression cast as a float.

Syntax

```
cast_float(exp)
```

cast_integer

Returns the value of the expression cast as a integer.

Syntax

```
cast_integer(exp)
```

cast_numeric

Returns the value of string_exp cast as a numeric value.

Syntax

```
cast_numeric(string_exp)
```

cast_real

Returns the value of the expression cast as a real.

Syntax

```
cast_real(exp)
```

cast_smallint

Returns the value of the expression cast as a smallint.

Syntax

```
cast_smallint(exp)
```

cast_varchar

Returns the value of the expression cast as a varchar.

Syntax

```
cast_varchar(exp)
```

MS Access Math

log

Returns the natural logarithm of numeric_exp.

Syntax

```
log(numeric_exp)
```

rand

Generates a random number using integer_exp as a seed value.

Syntax

```
rand(integer_exp)
```

MS Access Trigonometry

atan

Returns the arctangent of numeric_exp in radians. The arctangent is the angle whose tangent is numeric_exp.

Syntax

```
atan(numeric_exp)
```

cos

Returns the cosine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
cos(numeric_exp)
```

sin

Returns the sine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
sin(numeric_exp)
```

tan

Returns the tangent of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
tan(numeric_exp)
```

Oracle

add_months

Returns the datetime resulting from adding integer_exp months to date_exp.

Syntax

```
add_months ( date_exp, integer_exp )
```

ascii

Returns a number representing the ascii code value of the leftmost character of string_exp, e.g. ascii ('A') is 65.

Syntax

```
ascii ( string_exp )
```

ceil

Returns the smallest integer greater than or equal to numeric_exp.

Syntax

```
ceil ( numeric_exp )
```

char_length

Returns the number of characters in string_exp.

Syntax

```
char_length ( string_exp )
```

chr

Returns the character that has the ASCII code value specified by integer_exp. integer_exp should be between 0 and 255.

Syntax

```
chr ( integer_exp )
```

concat

Returns a string that is the result of concatenating string_exp1 to string_exp2.

Syntax

```
concat ( string_exp1, string_exp2 )
```

decode

DECODE compares expr to each search value one by one. If expr is equal to a search, then returns the corresponding result. If no match is found, then returns default. If default is omitted, then returns null.

Syntax

```
decode ( expr , search , result [, search , result]... [, default] )
```

dump

Returns internal representation of 'expr' with the format of numeric_exp1 starting from position numeric_exp2 for numeric_exp3.

Syntax

```
dump ( expr [ , numeric_exp1 [ , numeric_exp2 [ , numeric_exp3 ] ] ] )
```

greatest

Returns the greatest value in a list of expressions.

Syntax

```
greatest ( exp_list )
```

initcap

Returns string_exp, with the first letter of each word in uppercase, all other letters in lowercase. Words are delimited by white space or characters that are not alphanumeric.

Syntax

```
initcap ( string_exp )
```

instr

Searches string_exp1 from the integer_exp1 position for the (integer_exp2)th occurance of string_exp2. If integer_exp1 is negative then the search is backwards from the end of string_exp1. Returns an integer indicating the position of string_exp2.

Syntax

```
instr ( string_exp1, string_exp2 [ , integer_exp1 [ , integer_exp2 ] ] )
```

instrb

Searches string_exp1 from the integer_exp1 position for the (integer_exp2)th occurance of string_exp2. If integer_exp1 is negative then the search is backwards from the end of string_exp1. The result returned indicates the position (byte number) where search was found.

Syntax

```
instrb ( string_exp1, string_exp2 [ , integer_exp1 [ , integer_exp2 ] ] )
```

least

Returns the least value in a list of expressions.

Syntax

```
least ( exp_list )
```

length

Returns the number of characters in string_exp.

Syntax

```
length ( string_exp )
```

lengthb

Returns the number of bytes in string_exp.

Syntax

```
lengthb ( string_exp )
```

lpad

Returns string_exp1 padded to length integer_exp with occurrences of string_exp2. If string_exp1 is longer than integer_exp then returns the appropriate portion of string_exp1.

Syntax

```
lpad ( string_exp1, integer_exp [ , string_exp2 ] )
```

ltrim

Returns string_exp1, with leading characters removed up to the first character not in string_exp2, e.g. ltrim('xyxXxyAB', 'xy') returns 'XxyAB'.

Syntax

```
ltrim ( string_exp1 [ , string_exp2 ] )
```

months_between

Returns the number of months from date_exp1 to date_exp2. If date_exp1 is later than date_exp2 then the result will be a positive number. The days and time portion of the difference are ignored, i.e. the months are not rounded, except if date_exp1 and date_exp2 are the last days of a month.

Syntax

```
months_between ( date_exp1, date_exp2 )
```

new_time

Returns the Datetime in timezone 'new_tz' for 'datetime' in 'old_tz' timezone. 'Old_tz' and 'new_tz' can be one of 'AST', 'ADT', 'BST', 'BDT', 'CST', 'CDT', 'EST', 'EDT', 'HST', 'HDT', 'MST', 'MDT', 'NST', 'PST', 'PDT', 'YST' or 'YDT'.

Syntax

```
new_time ( datetime_exp, old_tz, new_tz )
```

next_day

Returns the datetime of the first weekday named by string_exp that is later than datetime_exp. The return value has the same hours, minutes, and seconds as datetime_exp.

Syntax

```
next_day ( datetime_exp, string_exp )
```

nls_initcap

Returns string_exp1 with the first letter of each word in uppercase, all other letters in lowercase. Words are delimited by white space or characters that are not alphanumeric. string_exp2 specifies the sorting sequence.

Syntax

```
nls_initcap ( string_exp1 [ , string_exp2 ] )
```

nls_lower

Returns string_exp1 with all letters in lowercase. string_exp2 specifies the sorting sequence.

Syntax

```
nls_lower ( string_exp1 [ , string_exp2 ] )
```

nls_upper

Returns string_exp1 with all letters in uppercase. string_exp2 specifies the sorting sequence.

Syntax

```
nls_upper ( string_exp1 [ , string_exp2 ] )
```

nvl

If exp is null (missing) returns constant. If exp is not null returns exp. Valid for numeric_exp, string_exp, date_exp, and time_exp.

Syntax

```
nvl ( exp, constant )
```

replace

Replaces all occurrences of string_exp2 in string_exp1 with string_exp3. If string_exp3 is not specified then it replaces all occurrences with null (ie: removes all occurances of string_exp2).

Syntax

```
replace ( string_exp1, string_exp2 [ , string_exp3 ] )
```

round

Returns numeric_exp rounded to the nearest value integer_exp places right of the decimal point. If integer_exp is negative, numeric_exp is rounded to the nearest value absolute (integer_exp) places to the left of the decimal point, e.g., round (125, -1) rounds to 130.

Syntax

```
round ( numeric_exp [ , integer_exp ] )
```

rpad

Returns string_exp1 right-padded to length integer_exp with occurrences of string_exp2. If string_exp1 is longer than integer_exp then returns the appropriate portion of string_exp1. If string_exp2 is not specified then spaces are used.

Syntax

```
rpad ( string_exp1, integer_exp [ , string_exp2 ] )
```

rtrim

Returns string_exp1, with final characters removed after the last character not in string_exp2, e.g. rtrim('ABxXxyx', 'xy') returns 'ABxX'. If string_exp2 is not specified it removes th final space characters.

Syntax

```
rtrim ( string_exp1 [ , string_exp2 ] )
```

sign

Returns an indicator of the sign of numeric_exp: +1 if numeric_exp is positive, 0 if zero or -1 if negative.

Syntax

```
sign ( numeric_exp )
```

soundex

Returns a character string containing the phonetic representation of string_exp.

Syntax

```
soundex ( string_exp )
```

substr

Returns the substring of string_exp that starts at position integer_exp1. The first character in string_exp is at position 1. integer_exp2 can be used to select fewer characters, by default it selects character to the end of the string.

Syntax

```
substr ( string_exp, integer_exp1 [ , integer_exp2 ] )
```

substrb

Same as substr, except that the arguments are expressed in bytes (not characters).

Syntax

```
substrb ( string_exp, numeric_exp1 [ , numeric_exp2 ] )
```

{sysdate}

Returns a datetime value representing the current date and time of the computer that the database software runs on.

Syntax

```
{ sysdate }
```

to_char

Returns the string representation of exp with the format of string_exp. exp can either be a date value or a numeric value.

Syntax

```
to_char ( exp [ , string_exp ] )
```

to_date

Converts string_exp1 to a datetime value as specified by the format string_exp2. string_exp3 specifies format elements such as language.

Syntax

```
to_date ( string_exp1 [ , string_exp2 [ , string_exp3 ] ] )
```

to_number

Converts string_exp1 to a numeric value as specified by the format string_exp2. string_exp3 specifies format elements such as currency information.

Syntax

```
to_number ( string_exp1, string_exp2, string_exp3 )
```

translate

Returns string_exp1, with all occurrences of each character in string_exp2 replaced by its corresponding character in string_exp3.

Syntax

```
translate ( string_exp1, string_exp2, string_exp3 )
```

trunc

Truncates the date_exp using the format specified by string_exp. For example, if string_exp is 'YEAR' then date_exp is truncated to the first day of the year.

Syntax

```
trunc ( date_exp, string_exp )
```

trunc

Truncates digits from numeric_exp1 using numeric_exp2 as the precision.

Syntax

```
trunc ( numeric_exp1, numeric_exp2 )
```

{user}

Returns the username of the current Oracle user.

Syntax

```
{ user }
```

vsize

Returns the number of bytes in the internal representation of 'exp'. 'exp' must be a string expression.

Syntax

```
vsize ( exp )
```

Oracle Math

log

Returns the logarithm of numeric_exp2 to the base numeric_exp1.

Syntax

```
log ( numeric_exp1, numeric_exp2 )
```

Oracle Trigonometry

acos

Returns the arccosine of numeric_exp in radians. The arccosine is the angle whose cosine is numeric_exp.

Syntax

```
acos ( numeric_exp )
```

asin

Returns the arcsine of numeric_exp in radians. The arcsine is the angle whose sine is numeric_exp.

Syntax

```
asin ( numeric_exp )
```

atan

Returns the arctangent of numeric_exp in radians. The arctangent is the angle whose tangent is numeric_exp.

Syntax

```
atan ( numeric_exp )
```

atan2

Returns the arctangent of the x and y coordinates specified by numeric_exp1 and numeric_exp2, respectively, in radians. The arctangent is the angle whose tangent is numeric_exp2 / numeric_exp1.

Syntax

```
atan2 ( numeric_exp1 ,numeric_exp2 )
```

cos

Returns the cosine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
cos ( numeric_exp )
```

cosh

Returns the hyperbolic cosine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
cosh ( numeric_exp )
```

sin

Returns the sine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
sin ( numeric_exp )
```

sinh

Returns the hyperbolic sine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
sinh ( numeric_exp )
```

tan

Returns the tangent of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
tan ( numeric_exp )
```

tanh

Returns the hyperbolic tangent of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
tanh ( numeric_exp )
```

Red Brick

ceil

Returns the smallest integer greater than or equal to numeric_exp or string_exp. Note that string_exp must represent a valid numeric value.

Syntax

```
ceil ( numeric_exp | string_exp )
```

concat

This function concatenates character strings and returns the concatenated string of characters.

Syntax

```
concat ( string_exp1 , string_exp2 )
```

{current_user}

Returns the database username (authorization ID) of the current user.

Syntax

```
{ current_user }
```

date

This function creates a date value from a character string or a timestamp expression and returns a date data type. The expression can be either characters or timestamp.

Syntax

```
date ( expression )
```

dateadd

This function adds an interval to a datetime value and returns a result that is the same datetime data type as that of datetime_expression. The datepart refers to year, month, day, hour, minute, second. The interval must be an integer and datetime_exp can be date, time or timestamp.

Syntax

```
dateadd ( { datepart }, interval, datetime_exp )
```

datediff

This function finds the difference between two datetime expressions and returns an integer result in datepart units. The datepart refers to year, month, day, hour, minute, second. The datetime_exp can be date, time or timestamp.

Syntax

```
datediff ( { datepart }, datetime_exp, datetime_exp )
```

datename

This function extracts the specified datepart component and returns its value as a character string. The datepart refers to year, month, day, hour, minute, second. The datetime_exp can be date, time or timestamp.

Syntax

```
datename ( { datepart }, datetime_exp )
```

dec

This function converts a specified value to a decimal value and returns a value with the data type decimal (precision, scale). The default value of precision is 9. The default value of scale is 0.

Syntax

```
dec ( expression, [precision, scale] )
```

decimal

This function converts a specified value to a decimal value and returns a value with the data type decimal (precision, scale). The default value of precision is 9. The default value of scale is 0.

Syntax

```
decimal ( expression, [precision, scale] )
```

decode

This function compares and converts an expression to another value. If the expression matches target, it is replaced by the corresponding replacement; otherwise the expression is replaced by default or by NULL if no default is specified. The expressions can be any data type and all expressions must be the same data type.

Syntax

```
decode ( expression, target, replacement [,default] )
```

float

This function converts a specified value into a double-precision floating-point value.

Syntax

```
float ( numeric_exp )
```

ifnull

This function tests an expression for missing values and replaces each one with a specified value. If expression is NULL, this function returns substitute; otherwise it returns the value of the expression. The expressions can be any data type and all expressions must be the same data type.

Syntax

```
ifnull ( expression , substitute )
```

int

This function converts a specified numeric string into an integer value and returns an integer value. If the argument is null, this function returns NULL.

Syntax

```
int ( numeric_exp )
```

integer

This function converts a specified numeric string into an integer value and returns an integer value. If the argument is null, this function returns NULL.

Syntax

```
integer ( numeric_exp )
```

length

If the argument is not null, this function returns an integer result specifying the number of characters in the string; otherwise the result is NULL.

Syntax

```
length ( string_exp )
```

lengthb

If the argument is not null, this function returns an integer result specifying the number of bytes in the string. If the argument is null, the result is NULL.

Syntax

```
lengthb ( string_exp )
```

ltrim

If the argument is not null, this function removes leading blanks from the character string; otherwise the result is NULL.

Syntax

```
ltrim ( string_exp )
```

nullif

This function returns NULL if both expressions have the same value. If the expressions have different values, the value of the first expression is returned. The exp1 and exp2 can be any data type and must be the same data type.

Syntax

```
nullif ( exp1, exp2 )
```

positionb

If the first string_exp is located, this function returns an integer that is relative to the beginning byte position of the first string_exp in the second string_exp. If the first string_exp is not located, the result is 0. If the first string_exp is of zero length, the result is 1. If the first string_exp is null, an error message is returned. If the second string_exp is null, the result is 0.

Syntax

```
positionb ( string-exp, string_exp )
```

real

This function returns a real value. If the argument is null, this function returns NULL.

Syntax

```
real ( numeric_exp )
```

round

Returns numeric_exp rounded to the nearest value integer_exp places right of the decimal point. If integer_exp is negative, numeric_exp is rounded to the nearest value absolute (integer_exp) places to the left of the decimal point, e.g., round (125, -1) rounds to 130.

Syntax

```
round ( numeric_exp, integer_exp )
```

rtrim

If the argument is not null, this function removes trailing blanks from the character string; otherwise the result is NULL.

Syntax

```
rtrim ( string_exp )
```

sign

This function calculates the sign of the expression, and returns 1 for a positive value, –1 for a negative value, and 0 for zero.

Syntax

```
sign ( numeric_exp )
```

string

This function converts numeric or datetime values to character strings. The expression can be numeric or datetime.

Syntax

```
string ( expression [, length [, scale]] )
```

substr

If the first argument is not null, this function returns the substring that begins at position start and continues for length characters. If length is not specified, this function returns a substring from start to the end of string_exp.

Syntax

```
substr ( string_exp, start_integer, length_integer )
```

substrb

If the first argument is not null, this function returns the substring that begins at position start and continues for length bytes. If length is not specified, this function returns a substring from start to the end of string_exp.

Syntax

```
substrb ( string_exp, start_integer, length_integer )
```

time

This function creates a time value from a character string or a time-stamp data type expression.

Syntax

```
time ( expression )
```

timestamp

This function creates a time-stamp value from a character string.

Syntax

```
timestamp ( timestamp_exp )
```

timestamp

This function creates a time-stamp value from time and date values. If there are two arguments, the first must be a date expression and the second must be a time expression, separated by a comma (,). If either the date expression or the time expression is null, the resulting time-stamp expression is also null.

Syntax

```
timestamp ( date_exp, time_exp )
```

to_char

This function is a datetime scalar function that operates on a date, time, or timestamp data type and returns the character string specified by a given format.

Syntax

```
to_char ( source_date, format_str )
```

SQL Server

ascii

Returns a number representing the ascii code value of the leftmost character of string_exp, e.g. ascii ('A') is 65.

Syntax

```
ascii(string_exp)
```

char

Returns the character that has the ASCII code value specified by integer_exp. integer_exp should be between 0 and 255. For example, char(65) has the value 'A'.

Syntax

```
char(integer_exp)
```

charindex

Searches string_exp2 for the first occurrence of string_exp1 and returns an integer. start_location is the character position to start searching for string_exp1 in string_exp2. If start_location is not given, is a negative number, or is zero, the search starts at the beginning of string_exp2.

Syntax

```
charindex ( string_exp1, string_exp2 [ , start_location ]  )
```

{current_user}

Returns the name of the current user.

Syntax

```
{ current_user }
```

datalength

Returns the length in bytes of the string.

Syntax

```
datalength(string_exp)
```

dateadd

Returns the date resulting from adding integer_exp units indicated by datepart(e.g. day, month, year) to date_exp.

Syntax

```
dateadd({datepart}, integer_exp, date_exp)
```

datediff

Returns the number of units indicated by datepart(e.g. day, month, year) between date_exp1 and date_exp2.

Syntax

```
datediff({datepart}, date_exp1, date_exp2)
```

datename

Returns part of a datetime, smalldatetime, date or time value as an ASCII string. Note that the datepart argument must be a keyword representing a datepart or its abbreviation recognized by Microsoft SQL Server and must be enclosed in curly brackets.

Syntax

```
datename ( ' { ' datepart ' } ' , date_exp )
```

Example

```
datename ( {mm}, 2000-01-01 )
```

Result: January.

datepart

Returns part of a datetime, smalldatetime, date or time value (for example, the month) as an integer. Note that the datepart argument must be a keyword representing a datepart or its abbreviation recognized by Microsoft SQL Server and must be enclosed in curly brackets.

Syntax

```
datepart ( ' { ' datepart ' } ' , date_exp )
```

Example

```
datepart ( {wk}, 2000-01-01 )
```

Result: 1 (first week of the year).

day

Returns the day portion of date_exp. Same as extract(day from date_exp).

Syntax

```
day(date_exp)
```

difference

Returns an integer value representing the difference between the values returned by the data source_specific soundex function for string_exp1 and string_exp2. The value returned ranges from 0 to 4, with 4 indicating the best match. Note that 4 does not mean that the strings are equal.

Syntax

```
difference(string_exp1, string_exp2)
```

getdate

Returns a datetime value representing the current date and time of the computer that the database software runs on.

Syntax

```
getdate()
```

left

Returns the leftmost integer_exp characters of string_exp.

Syntax

```
left(string_exp, integer_exp)
```

ltrim

Returns string_exp with leading spaces removed.

Syntax

```
ltrim(string_exp)
```

month

Returns the month portion of date_exp. Same as extract(month from date_exp).

Syntax

```
month(date_exp)
```

patindex

Returns an integer which represents the starting position if the first occurrence of string_exp1 is in the string_exp2; returns 0 if string-exp1 is not found. The % wildcard character must precede and follow pattern in string_exp1, except when searching for first or last characters.

Syntax

```
patindex ( string_exp1, string_exp2 )
```

Example

```
patindex ( '%nos%', 'Cognos' )
```

Result: 4

replace

Replaces all occurrences of string_exp2 in string_exp1 with string_exp3.

Syntax

```
replace ( string_exp1 , string_exp2 , string_exp3 )
```

replicate

Returns a string consisting of string_exp repeated integer_exp times.

Syntax

```
replicate(string_exp, integer_exp)
```

reverse

Returns the reverse of the character expression.

Syntax

```
reverse ( string_exp )
```

right

Returns the rightmost integer_exp characters of string_exp.

Syntax

```
right(string_exp, integer_exp)
```

round

Returns numeric_exp rounded to the nearest value integer_exp places right of the decimal point.

Syntax

```
round(numeric_exp,integer_exp)
```

rtrim

Returns string_exp with trailing spaces removed.

Syntax

```
rtrim(string_exp)
```

sign

Returns an indicator of the sign of numeric_exp: +1 if numeric_exp is positive, 0 if zero or -1 if negative.

Syntax

```
sign(numeric_exp)
```

soundex

Returns a four character string representing the sound of the words in string_exp.

Syntax

```
soundex(string_exp)
```

space

Returns a string consisting of integer_exp spaces.

Syntax

```
space(integer_exp)
```

str

Returns a string representation of numeric_exp. integer_exp1 is the length of the string returned. integer_exp2 is the number of decimal digits.

Syntax

```
str(numeric_exp [ , integer_exp1 [ , integer_exp2 ] ] )
```

stuff

Returns a string where length (integer_exp2) characters have been deleted from string_exp1 beginning at start (integer_exp1) and where string_exp2 has been inserted into string_exp1 at start. The first character in a string is at position 1.

Syntax

```
stuff(string_exp1, integer_exp1, integer_exp2, string_exp2)
```

year

Returns the year portion of date_exp. Same as extract(year from date_exp).

Syntax

```
year(date_exp)
```

SQL Server Cast

cast_char

Returns the value of the expression cast as a char. A limit of 30 characters is returned.

Syntax

```
cast_char(exp)
```

cast_float

Returns the value of the expression cast as a float.

Syntax

```
cast_float(exp)
```

cast_integer

Returns the value of the expression cast as an integer.

Syntax

```
cast_integer(exp)
```

cast_real

Returns the value of the expression cast as a real.

Syntax

```
cast_real(exp)
```

cast_smallint

Returns the value of the expression cast as a small integer.

Syntax

```
cast_smallint(exp)
```

SQL Server Math

log

Returns the natural logarithm of numeric_exp.

Syntax

```
log(numeric_exp)
```

log10

Returns the base ten logarithm of numeric_exp.

Syntax

```
log10(numeric_exp)
```

pi

Returns the constant value of pi as a floating point value.

Syntax

```
pi()
```

rand

Generates a random number using integer_exp as a seed value.

Syntax

```
rand(integer_exp)
```

SQL Server Trigonometry

acos

Returns the arccosine of numeric_exp in radians. The arccosine is the angle whose cosine is numeric_exp.

Syntax

```
acos(numeric_exp)
```

asin

Returns the arcsine of numeric_exp in radians. The arcsine is the angle whose sine is numeric_exp.

Syntax

```
asin(numeric_exp)
```

atan

Returns the arctangent of numeric_exp in radians. The arctangent is the angle whose tangent is numeric_exp.

Syntax

```
atan(numeric_exp)
```

atn2

Returns the arctangent of the x and y coordinates specified by numeric_exp1 and numeric_exp2, respectively, in radians. The arctangent is the angle whose tangent is numeric_exp1.

Syntax

```
atn2(numeric_exp1, numeric_exp2)
```

cos

Returns the cosine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
cos(numeric_exp)
```

cot

Returns the cotangent of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
cot(numeric_exp)
```

degrees

Returns numeric_exp radians converted to degrees.

Syntax

```
degrees(numeric_exp)
```

radians

Returns the number of radians converted from numeric_exp degrees.

Syntax

```
radians(numeric_exp)
```

sin

Returns the sine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
sin(numeric_exp)
```

tan

Returns the tangent of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
tan(numeric_exp)
```

Teradata

account

This function returns the account string for the current user.

Syntax

```
{account}
```

add_months

Returns the date or the datetime resulting from adding integer_exp months to date_exp or datetime_exp.

Syntax

```
add_months ( date_exp | datetime_exp, integer_exp )
```

bytes

This function returns the number of bytes contained in the specified byte string.The byte_exp are restricted to BYTE or VARBYTE.

Syntax

```
bytes ( byte_exp )
```

case_n

This function evaluates a list of conditions and returns the position of the first condition that evaluates to TRUE, provided that no prior condition in the list evaluates to UNKNOWN. The keywords must be enclosed in curly brackets. The NO CASE is an optional condition that evaluates to TRUE if every conditional_expression in the list evaluates to FALSE. The NO CASE OR UNKNOWN condition evaluates to TRUE if every conditional_expression in the list evaluates to FALSE, or if a conditional_expression evaluates to UNKNOWN and all prior conditions in the list evaluate to FALSE. The UNKNOWN is an optional condition that evaluates to TRUE if a conditional_expression evaluates to UNKNOWN and all prior conditions in the list evaluate to FALSE.

Syntax

```
case_n ( condition_exp_list [, NO CASE | UNKNOWN | NO CASE OR UNKNOWN [, UNKNOWN
 ]] )
```

char2hexint

This function returns the hexadecimal representation for a character string.

Syntax

```
char2hexint ( string_exp )
```

characters

This function returns an integer value representing the number of logical characters or bytes contained in the specified operand string.

Syntax

```
characters ( string_exp )
```

database

This function returns the name of the default database for the current user.

Syntax

```
{database}
```

date

This function returns the current date.

Syntax

```
{date}
```

format

This function returns the declared format for the named expression. The data type returned by a FORMAT phrase is a variable character string of up to 30 characters.

Syntax

```
format ( expression )
```

index

This function returns the position in string_exp1 where string_exp2 starts.

Syntax

```
index ( string_exp1, string_exp2 )
```

log

Computes the base 10 logarithm of an argument. The numeric_exp is a non-zero, positive numeric expression.

Syntax

```
log ( numeric_exp )
```

nullif

This function returns NULL if scalar_exp1 and scalar_exp2 are equal. Otherwise, it returns its first argument, scalar_exp1. The scalar_exp1 and scalar_exp2 can be any data type.

Syntax

```
nullif ( scalar_exp1, scalar_exp2 )
```

nullifzero

This function converts data from zero to null to avoid problems with division by zero.

Syntax

```
nullifzero ( numeric_exp )
```

profile

This function returns the current profile for the session or NULL if none.

Syntax

```
{profile}
```

random

This function returns a random integer number for each row of the results table. The lower_bound and upper_bound are integer constants. The limits for lower_bound, upper_bound range from -2147483648 to 2147483647, inclusive. The upper_bound must be greater than or equal to lower_bound.

Syntax

```
random ( lower_bound, upper_bound )
```

role

This function returns the current role for the session or NULL if none.

Syntax

```
{role}
```

session

This function returns the number of the session for the current user.

Syntax

```
{session}
```

soundex

This function returns a character string that represents the Soundex code for string_exp.

Syntax

```
soundex ( string_exp )
```

substr

Returns the substring of string_exp that starts at position integer_exp1. The first character in string_exp is at position 1. integer_exp2 can be used to select fewer characters, by default it selects character to the end of the string.

Syntax

```
substr ( string_exp, integer_exp1 [ , integer_exp2 ] )
```

time

This function returns the current time based on a 24-hour day.

Syntax

```
{time}
```

type

This function returns the data type defined for an expression.

Syntax

```
type ( expression )
```

user

This function returns the user name of the current user.

Syntax

```
{user}
```

vargraphic

This function returns a character string that represents the vargraphic code for string_exp.

Syntax

```
vargraphic ( string_exp )
```

zeroifnull

This function converts data from null to 0 to avoid cases where a null result creates an error. If the numeric_exp is not null, it returns the value of the numeric_exp, if numeric_exp is a character string, it is converted to a numeric value of FLOAT data type. If the numeric_exp is null or zero, it returns zero.

Syntax

```
zeroifnull ( numeric_exp )
```

Teradata Trigonometry

acos

Returns the arccosine of numeric_exp in radians. The arccosine is the angle whose cosine is numeric_exp. The values of numeric_exp must be between -1 and 1, inclusive.

Syntax

```
acos ( numeric_exp )
```

acosh

Returns the inverse hyperbolic cosine of an argument. The numeric_exp can be any real number equal to or greater than 1.

Syntax

```
acosh ( numeric_exp )
```

asin

Returns the arcsine of numeric_exp in radians. The arcsine is the angle whose sine is numeric_exp. The values of numeric_exp must be between -1 and 1, inclusive.

Syntax

```
asin ( numeric_exp )
```

asinh

Returns the inverse hyperbolic sine of an argument. The numeric_exp can be any real number.

Syntax

```
asinh ( numeric_exp )
```

atan

Returns the arctangent of numeric_exp in radians. The arctangent is the angle whose tangent is numeric_exp.

Syntax

```
atan ( numeric_exp )
```

atan2

Returns the arctangent of the x and y coordinates specified by numeric_exp1 and numeric_exp2, respectively, in radians. ATAN2(x,y) equals ATAN(y/x), except that x can be 0 in ATAN2(x,y). The returned angle is between - and π radians, excluding π.

Syntax

```
atan2 ( numeric_exp1, numeric_exp2 )
```

atanh

Returns the inverse hyperbolic tangent of an argument. The numeric_exp can be any real number between 1 and -1, excluding 1 and -1.

Syntax

```
atanh (numeric_exp )
```

cos

Returns the cosine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
cos ( numeric_exp )
```

cosh

Returns the hyperbolic cosine of an argument. The numeric_exp can be any real number.

Syntax

```
cosh ( numeric_exp )
```

sin

Returns the sine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
sin ( numeric_exp )
```

sinh

Returns the hyperbolic sine of an argument. The numeric_exp can be any real number.

Syntax

```
sinh ( numeric_exp )
```

tan

Returns the tangent of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
tan ( numeric_exp )
```

tanh

Returns the hyperbolic tangent of an argument. The numeric_exp can be any real number.

Syntax

```
tanh ( numeric_exp )
```

SAP BW

SAP BW Trigonometry

arccos

Returns the arccosine of numeric_exp in radians. The arccosine is the angle whose cosine is numeric_exp.

Syntax

```
arccos ( numeric_exp )
```

arcsin

Returns the arcsine of numeric_exp in radians. The arcsine is the angle whose sine is numeric_exp.

Syntax

```
arcsin ( numeric_exp )
```

arctan

Returns the arctangent of numeric_exp in radians. The arctangent is the angle whose tangent is numeric_exp.

Syntax

```
arctan ( numeric_exp )
```

cos

Returns the cosine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
cos ( numeric_exp )
```

sin

Returns the sine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
sin ( numeric_exp )
```

tan

Returns the tangent of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
tan ( numeric_exp )
```

coshyp

Returns the hyperbolic cosine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
coshyp ( numeric_exp )
```

sinhyp

Returns the hyperbolic sine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
sinhyp ( numeric_exp )
```

tanhyp

Returns the hyperbolic tangent of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
tanhyp ( numeric_exp )
```

SAP BW Math

log10

Returns the base ten logarithm of numeric_exp.

Syntax

```
log10 ( numeric_exp )
```

Sybase

ascii

Returns a number representing the ascii code value of the leftmost character of string_exp.

Syntax

```
ascii ( string_exp )
```

Example

```
ascii( 'A' )
```

Result: 65

char

Converts a single-byte integer value to a character value. char is usually used as the inverse of ascii. integer_exp must be between 0 and 255. Returns a char datatype. If the resulting value is the first byte of a multibyte character, the character may be undefined.

Syntax

```
char ( integer_exp )
```

charindex

Searches string_exp2 for the first occurrence of string_exp1and returns an integer, which represents its starting position. If string_exp1 is not found, it returns 0. If string_exp1 contains wildcard characters, charindex treats it them as literals.

Syntax

```
charindex ( string_exp1, string_exp2 )
```

datalength

Returns the length in bytes of the string.

Syntax

```
datalength ( string_exp )
```

dateadd

Returns the date resulting from adding integer_exp units indicated by datepart(e.g. day, month, year) to date_exp. Note that the datepart argument must be enclosed in curly brackets.

Syntax

```
dateadd ( ' { ' datepart ' } ' , integer_exp, date_exp )
```

Example

```
dateadd ( {dd}, 16, 1997-06-16 )
```

Result: Jul 2, 1997

datediff

Returns the number of units indicated by datepart(e.g. day, month, year) between date_exp1 and date_exp2. Note that the datepart argument must be enclosed in curly brackets.

Syntax

```
datediff ( ' { ' datepart ' } ' , date_exp1, date_exp2 )
```

Example

```
datediff ( {yy}, 1984-01-01, 1997-01-01 )
```

Result: 13

datename

Returns part of a datetime, smalldatetime, date or time value as an ASCII string. Note that the datepart argument must be enclosed in curly brackets.

Syntax

```
datename ( ' { ' datepart ' } ' , date_exp )
```

Example

```
datename ( {mm}, 1999-05-01 )
```

Result: May

datepart

Returns part of a datetime, smalldatetime, date or time value (for example, the month) as an integer. Note that the datepart argument must be enclosed in curly brackets.

Syntax

```
datepart ( ' { ' datepart ' } ' , date_exp )
```

Example

```
datepart ( {mm}, 1999-05-01 )
```

Result: 5

day

Returns the day of the month (1-31) from date_exp.

Syntax

```
day ( date_exp )
```

difference

Returns an integer value representing the difference between the values returned by the data source_specific soundex function for string_exp1 and string_exp2. The value returned ranges from 0 to 4, with 4 indicating the best match. Note that 4 does not mean that the strings are equal.

Syntax

```
difference ( string_exp1, string_exp2 )
```

getdate

Returns current system date and time.

Syntax

```
getdate ()
```

left

Returns the leftmost integer_exp characters of string_exp.

Syntax

```
left ( string_exp, integer_exp )
```

ltrim

Returns string_exp with leading spaces removed.

Syntax

```
ltrim ( string_exp )
```

month

Returns the month (an integer from 1-12) from date_exp.

Syntax

```
month ( date_exp )
```

patindex

Returns an integer which represents the starting position if the first occurrence of string_exp1 is in the string_exp2; returns 0 if string-exp1 is not found. By default, patindex returns the offset in characters. To return offset in bytes, that is, multibyte character string, specify using bytes. The % wildcard character must precede and follow pattern in string_exp1, except when searching for first or last characters

Syntax

```
patindex ( string_exp1, string_exp2 [ using {bytes | chars | characters} ] )
```

rand

Returns a random float value between 0 and 1, using the optional integer as a seed value.

Syntax

```
rand ( integer_exp )
```

replicate

Returns a string with the same datatype as string_exp, containing the same expression repeated integer_exp times or as many times as will fit into a 225-byte space, whichever is less.

Syntax

```
replicate ( string_exp, integer_exp )
```

reverse

Returns the reverse of the character or binary expression; if string_exp is "abcd", it returns "dcba".

Syntax

```
reverse ( string_exp )
```

right

Returns the rightmost integer_exp characters of string_exp.

Syntax

```
right ( string_exp, integer_exp )
```

round

Returns numeric_exp rounded to the nearest value integer_exp places right of the decimal point.

Syntax

```
round ( numeric_exp, integer_exp )
```

rtrim

Returns string_exp with trailing spaces removed.

Syntax

```
rtrim ( string_exp )
```

soundex

Returns a four-character soundex code for character strings that are composed of a contigous sequence of valid single- or double byte Roman letter.

Syntax

```
soundex ( string_exp )
```

space

Returns a string with the indicated number of single-byte space.

Syntax

```
space ( integer_exp )
```

str

Returns a string representation of numeric_exp. integer_exp1 is the length of the string returned. integer_exp2 is the number of decimal digits. length and decimal are optional(default length is 10; default decimal is 0.)

Syntax

```
str ( numeric_exp [ , integer_exp1 [ , integer_exp2 ] ] )
```

stuff

Delete integer_exp2 characters from string_exp1 at integer_exp1, and then insert string_exp2 into string_exp1 at integer_exp. To delete characters without inserting other characters, string_exp2 should be NULL, not " ", which indicates a single space.

Syntax

```
stuff ( string_exp1, integer_exp1, integer_exp2, string_exp2 )
```

substring

Returns the substring of string_exp that starts at position integer_exp1. integer_exp2 specifies the number of characters in the substring.

Syntax

```
substring ( string_exp, integer_exp1, integer_exp2 )
```

to_unichar

Returns a unichar expression having the value of the integer_exp. If the integer_exp is in the range 0xD800..0xDFFF, the operation is aborted. If the integer_exp is in the range 0..0xFFFF, a single Unicode value is returned. If the integer_exp is in the range 0x10000..0x10FFFF, a surrogate pair is returned.

Syntax

```
to_unichar ( integer_exp )
```

uhighsurr

Returns 1 if the Unicode value at integer_exp is the high half of a surrogate pair (which should appear first in the pair). Otherwise, returns 0. This function allows you to write explicit code for surrogate handling. Particularly, if a substring starts on a Unicode character where uhighsurr() is true, extract a substring of at least 2 Unicode values, as substr() does not extract just 1. substr() does not extract half of a surrogate pair.

Syntax

```
uhighsurr ( string_exp, integer_exp )
```

ulowsurr

Returns 1 if the Unicode value at integer_exp is the low half of a surrogate pair (which should appear second in the pair). Otherwise, returns 0. This function allows you to explicitly code around the adjustments performed by substr(), stuff(), and right(). Particularly, if a substring ends on a Unicode value where ulowsurr() is true, extract a substring of 1 less characters (or 1 more), since substr() does not extract a string that contains an unmatched surrogate pair.

Syntax

```
ulowsurr ( string_exp, integer_exp )
```

uscalar

Returns the Unicode scalar value for the first Unicode character in string_exp. If the first character is not the high-order half of a surrogate pair, then the value is in the range 0..0xFFFF. If the first character is the high-order half of a surrogate pair, a second value must be a low-order half, and the return value is in the range 0x10000..0x10FFFF. If this function is called on a uchar_expr containing an unmatched surrogate half and the operation aborted.

Syntax

```
uscalar ( string_exp )
```

year

Returns the year from date_exp.

Syntax

```
year ( date_exp )
```

Sybase Math

log

Returns the natural logarithm of numeric_exp.

Syntax

```
log ( numeric_exp )
```

log10

Returns the base ten logarithm of numeric_exp.

Syntax

```
log10 ( numeric_exp )
```

pi

Returns the constant value of pi as a floating point value.

Syntax

```
pi ()
```

sign

Returns an indicator of the sign of numeric_exp: +1 if numeric_exp is positive, 0 if zero or -1 if negative.

Syntax

```
sign ( numeric_exp )
```

Sybase Trigonometry

acos

Returns the arccosine of numeric_exp in radians. The arccosine is the angle whose cosine is numeric_exp.

Syntax

```
acos ( numeric_exp )
```

asin

Returns the arcsine of numeric_exp in radians. The arcsine is the angle whose sine is numeric_exp.

Syntax

```
asin ( numeric_exp )
```

atan

Returns the arctangent of numeric_exp in radians. The arctangent is the angle whose tangent is numeric_exp.

Syntax

```
atan ( numeric_exp )
```

tan

Returns the tangent of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
tan ( numeric_exp )
```

atn2

Returns the angle (in radians) whose tangent is (numeric_exp1/numeric_exp2).

Syntax

```
atn2 ( numeric_exp1, numeric_exp2 )
```

cos

Returns the cosine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
cos ( numeric_exp )
```

cot

Returns the cotangent of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
cot ( numeric_exp )
```

degrees

Returns numeric_exp radians converted to degrees.

Syntax

```
degrees ( numeric_exp )
```

radians

Return the degree equivalent of numeric_exp. Results are of the same type as numeric. For expressions of type numeric or decimal, the results have an internal precision of 77 and a scale equal to that of

the numeric expression. When the money datatype is used, internal conversion to float may cause loss of precision.

Syntax

```
radians ( numeric_exp )
```

sin

Returns the sine of numeric_exp where numeric_exp is an angle expressed in radians.

Syntax

```
sin ( numeric_exp )
```

Report Functions

_add_days

Returns the datetime resulting from adding integer_exp days to timestamp_exp.

Syntax

```
_add_days(timestamp_exp, integer_exp)
```

_add_months

Returns the datetime resulting from adding integer_exp months to timestamp_exp.

Syntax

```
_add_months(timestamp_exp, integer_exp)
```

_add_years

Returns the datetime resulting from adding integer_exp years to timestamp_exp.

Syntax

```
_add_years(timestamp_exp, integer_exp)
```

_age

Returns a number that is obtained from subtracting timestamp_exp from today's date in YYYYMMDD format (years, months, days).

Syntax

```
_age(timestamp_exp)
```

_day_of_week

Returns the day of week (between 1 and 7), where 1 is the first day of the week as indicated by integer_exp (between 1 and 7, 1 being Monday and 7 being Sunday). Note that in ISO 8601 standard, a week begins with Monday being day 1. In North America where Sunday is the first day of the week being day 7.

Syntax

```
_day_of_week(timestamp_exp, integer_exp)
```

Example

```
_day_of_week(2003-01-01, 1) will return 3 because 2003-01-01 was a Wednesday.
```

Result: 3

_day_of_year

Returns the ordinal for the day of the year in date_ exp (1 to 366). Also known as Julian day.

Syntax

```
_day_of_year(timestamp_exp)
```

_days_between

Returns a positive or negative number representing the number of days between the two datetime expressions. If timestamp_exp1 < timestamp_exp2 then the result will be a negative number.

Syntax

```
_days_between(timestamp_exp1, timestamp_exp2)
```

_days_to_end_of_month

Returns a number representing the number of days remaining in the month represented by the datetime expression timestamp_exp.

Syntax

```
_days_to_end_of_month(timestamp_exp)
```

_first_of_month

Returns a datetime that is the first day of the month represented by timestamp_exp.

Syntax

```
_first_of_month(timestamp_exp)
```

_last_of_month

Returns a datetime that is the last day of the month represented by timestamp_exp.

Syntax

```
_last_of_month(timestamp_exp)
```

_make_timestamp

Returns a timestamp constructed from integer_exp1 (the year), integer_exp2 (the month) and integer_exp3 (the day). The time portion defaults to 00:00:00.000 .

Syntax

```
_make_timestamp(integer_exp1, integer_exp2, integer_exp3)
```

_months_between

Returns a positive or negative number representing the number of months between the two datetime expressions. If timestamp_exp1 < timestamp_exp2 then the result will be a negative number.

Syntax

```
_months_between(timestamp_exp1, timestamp_exp2)
```

_week_of_year

Returns the number of the week of the year of the timestamp_exp according to the ISO 8601, in which week 1 of the year is the first week of the year to contain a Thursday, which is equivalent to the first week containing January 4th. A week starts on Monday (day1) and ends on Sunday (day7).

Syntax

```
_week_of_year(timestamp_exp)
```

_years_between

Returns a positive or negative integer number representing the number of years from timestamp_exp1 to timestamp_exp2. If timestamp_exp1 < timestamp_exp2 then a negative value is returned.

Syntax

```
_years_between(timestamp_exp1, timestamp_exp2)
```

_ymdint_between

Returns a number representing the difference between the datetime expressions timestamp_exp1 and timestamp_exp2. This value has the form YYYYMMDD, where YYYY represents the number of years, MM represents the number of months, and DD represents the number of days.

Syntax

```
_ymdint_between(timestamp_exp1, timestamp_exp2)
```

abs

Returns the absolute value of numeric_exp. The sign of negative values is changed to positive.

Syntax

```
abs(numeric_exp)
```

AsOfDate

Returns the date value of the As Of Time expression, if it is defined. Otherwise, AsOfDate returns the report execution date.

Syntax

```
AsOfDate()
```

AsOfTime

Returns the time value of the As Of Time expression, if it is defined. Otherwise, AsOfTime returns the report execution time.

Syntax

```
AsOfTime()
```

BurstKey

Returns burst key.

Syntax

```
BurstKey()
```

BurstRecipients

Returns the distribution list of burst recipients.

Syntax

```
BurstRecipients()
```

ceiling

Returns the smallest integer greater than or equal to numeric_exp.

Syntax

```
ceiling(numeric_exp)
```

CellValue

Returns the value of the current crosstab cell.

Syntax

```
CellValue()
```

character_length

Returns the number of characters in string_exp.

Syntax

```
character_length(string_exp)
```

ColumnNumber

Returns the current column number.

Syntax

```
ColumnNumber()
```

CubeCreatedOn

Returns the date and time when the cube was created. The dimension parameter specifies from which cube to retrieve the metadata. If the dimension source is a Cognos PowerCube (.mdc), the function returns a blank string. The initial creation date of a PowerCube is not maintained.

Syntax

```
CubeCreatedOn(dimension)
```

CubeCurrentPeriod

Returns the current period for the cube. The dimension parameter specifies from which cube to retrieve the metadata.

Syntax

```
CubeCurrentPeriod(dimension)
```

CubeDataUpdatedOn

Returns the date time that data in the cube was last updated. The dimension parameter specifies from which cube to retrieve the metadata.

Syntax

```
CubeDataUpdatedOn(dimension)
```

CubeDefaultMeasure

Returns the name of the default measure for the cube. The dimension parameter specifies from which cube to retrieve the metadata.

Syntax

```
CubeDefaultMeasure(dimension)
```

CubeDescription

Returns the description of the cube. The dimension parameter specifies from which cube to retrieve the metadata.

Syntax

```
CubeDescription(dimension)
```

CubeIsOptimized

Returns true if the cube is optimized. The dimension parameter specifies from which cube to retrieve the metadata.

Syntax

```
CubeIsOptimized(dimension)
```

CubeName

Returns the name of the cube. The dimension parameter specifies from which cube to retrieve the metadata.

Syntax

```
CubeName(dimension)
```

CubeSchemaUpdatedOn

Returns the date time that cube schema was last updated. The dimension parameter specifies from which cube to retrieve the metadata.

Syntax

```
CubeSchemaUpdatedOn(dimension)
```

exp

Returns e raised to the power of numeric_exp. The constant e is the base of the natural logarithm.

Syntax

```
exp(numeric_exp)
```

extract

Returns an integer representing the value of datepart_exp ("year", "month", "day", "hour", "minute", "second" (=default)) in datetime_exp.

Syntax

```
extract(datepart_exp , datetime_exp)
```

floor

Returns the largest integer less than or equal to numeric_exp.

Syntax

```
floor(numeric_exp)
```

GetLocale

Returns run locale (deprecated).

Syntax

```
GetLocale()
```

HorizontalPageCount

Returns current horizontal page count.

Syntax

```
HorizontalPageCount()
```

HorizontalPageNumber

Returns current horizontal page number.

Syntax

```
HorizontalPageNumber()
```

InScope

Returns boolean 1 (TRUE) when cell is in the scope of the data items and MUNs; otherwise, 0 (FALSE).

Syntax

```
InScope(dataItem, MUN, ...)
```

IsBursting

Returns boolean 1 (TRUE) when report will be distributed to given recipient; otherwise, 0 (FALSE) .

Syntax

```
IsBursting('recipientName')
```

IsCrosstabColumnNodeMember

Returns boolean 1 (TRUE) if the current node is a crosstab column node member.

Syntax

```
IsCrosstabColumnNodeMember()
```

IsCrosstabRowNodeMember

Returns boolean 1 (TRUE) if the current node is a crosstab row node member.

Syntax

```
IsCrosstabRowNodeMember()
```

IsFirstColumn

Returns boolean 1 (TRUE) if the current column is the first column.

Syntax

```
IsFirstColumn()
```

IsInnerMostCrosstabColumnNodeMember

Returns boolean 1 (TRUE) if the current node is an innermost crosstab column node member.

Syntax

```
IsInnerMostCrosstabColumnNodeMember()
```

IsInnerMostCrosstabRowNodeMember

Returns boolean 1 (TRUE) if the current node is an innermost crosstab row node member.

Syntax

```
IsInnerMostCrosstabRowNodeMember()
```

IsLastColumn

Returns boolean 1 (TRUE) if the current column is the last column.

Syntax

```
IsLastColumn()
```

IsLastInnerMostCrosstabColumnNodeMember

Returns boolean 1 (TRUE) if the current node is the last innermost crosstab column node member.

Syntax

```
IsLastInnerMostCrosstabColumnNodeMember()
```

IsLastInnerMostCrosstabRowNodeMember

Returns boolean 1 (TRUE) if the current node is the last innermost crosstab row node member.

Syntax

```
IsLastInnerMostCrosstabRowNodeMember()
```

IsOuterMostCrosstabColumnNodeMember

Returns boolean 1 (TRUE) if the current node is an outermost crosstab column node member.

Syntax

```
IsOuterMostCrosstabColumnNodeMember()
```

IsOuterMostCrosstabRowNodeMember

Returns boolean 1 (TRUE) if the current node is an outermost crosstab row node member.

Syntax

```
IsOuterMostCrosstabRowNodeMember()
```

IsPageCountAvailable

Returns boolean 1 (TRUE) if the page count is available for the current execution of the report.

Syntax

```
IsPageCountAvailable()
```

ln

Returns the natural logarithm of the numeric_exp.

Syntax

```
ln(numeric_exp)
```

Locale

Returns run locale.

Syntax

```
Locale()
```

lower

Returns string_exp with all uppercase characters shifted to lowercase.

Syntax

```
lower(string_exp)
```

mapNumberToLetter

The user should supply the initial letter and the number is added to it.

Syntax

```
mapNumberToLetter(string_exp, integer_exp)
```

Example

```
mapNumberToLetter ('a', 1) will result in 'b'.
```

Result: `'b'`

mod

Returns the integer value of the remainder (modulo) when integer_exp1 is divided by integer_exp2.

Syntax

```
mod (integer_exp1, integer_exp2)
```

ModelPath

Returns model path.

Syntax

```
ModelPath()
```

Now

Returns current system time.

Syntax

```
Now()
```

nullif

Returns NULL if string_exp1 equals string_exp2 (case insensitive), otherwise returns string_exp1.

Syntax

```
nullif(string_exp1, string_exp2)
```

octet_length

Returns the number of bytes in string_exp.

Syntax

```
octet_length(string_exp)
```

PageCount

Returns the current page count. If you run the report, this function works only when the report output is PDF or Excel. If you save the report output, this function works for all formats.

Syntax

```
PageCount()
```

PageName

Returns the current page name.

Syntax

```
PageName()
```

PageNumber

Returns current page number.

Syntax

```
PageNumber()
```

ParamCount

Returns parameter count of the variable identified by parameterName.

Syntax

```
ParamCount('parameterName')
```

ParamDisplayValue

Returns a string that is the parameter display value of the variable identified by parameterName.

Syntax

```
ParamDisplayValue('parameterName')
```

ParamName

Returns parameter name of the variable identified by parameterName.

Syntax

```
ParamName('parameterName')
```

ParamNames

Returns all parameter names.

Syntax

```
ParamNames()
```

ParamValue

Returns a string that is the parameter value of the variable identified by parameterName.

Syntax

```
ParamValue('parameterName')
```

position

Returns the integer value representing the starting position of string_exp1 in string_exp2 or 0 when string_exp1 is not found.

Syntax

```
position(string_exp1, string_exp2)
```

power

Returns num_exp1 raised to the power of num_exp2.

Syntax

```
power(num_exp1, num_exp2)
```

ReportAuthorLocale

Returns author locale.

Syntax

```
ReportAuthorLocale()
```

ReportCreateDate

Returns the date when the report was created.

Syntax

```
ReportCreateDate()
```

ReportDate

Returns report execution date and time.

Syntax

```
ReportDate()
```

ReportDescription

Returns report description. This function works only when the report is run from Cognos Connection.

Syntax

```
ReportDescription()
```

ReportID

Returns the report id.

Syntax

```
ReportID()
```

ReportLocale

Returns run locale.

Syntax

```
ReportLocale()
```

ReportName

Returns report name. This function works only when the report is run from Cognos Connection.

Syntax

```
ReportName()
```

ReportOption

Returns the value of run option variable identified by optionName. Possible values for optionName: attachmentEncoding, burst, cssURL, email, emailAsAttachment, emailAsURL, emailBody, emailSubject, emailTo, emailToAddress, history, metadataModel, outputEncapsulation, outputFormat, outputLocale, outputPageDefinition, outputPageOrientation, primaryWaitThreshold, print, printer, printerAddress, prompt, promptFormat, saveAs, saveOutput, secondaryWaitThreshold, verticalElements, xslURL.

Syntax

```
ReportOption('optionName')
```

ReportOutput

Returns the name of the output format. Possible return values: CSV, HTML, layoutDataXML, MHT, PDF, rawXML, singleXLS, spreadsheetML, XLS, XML, XLWA.

Syntax

```
ReportOutput()
```

ReportPath

Returns report path. This function works only when the report is run from Cognos Connection.

Syntax

```
ReportPath()
```

ReportProductLocale

Returns product locale.

Syntax

```
ReportProductLocale()
```

ReportSaveDate

Returns the date when the report was last saved.

Syntax

```
ReportSaveDate()
```

round

Returns numeric_exp rounded to the nearest value integer_exp places right of the decimal point. If integer_exp is negative, numeric_exp is rounded to the nearest value absolute (integer_exp) places to the left of the decimal point.

Syntax

```
round(numeric_exp, integer_exp)
```

Example

```
round (125, -1) rounds to 130.
```

Result: `130.`

RowNumber

Returns current row.

Syntax

```
RowNumber()
```

ServerLocale

Returns the locale of the server that runs the report.

Syntax

```
ServerLocale()
```

ServerName

Returns the name of the server that runs the report.

Syntax

```
ServerName()
```

sqrt

Returns the square root of numeric_exp. numeric_exp must be non-negative.

Syntax

```
sqrt(numeric_exp)
```

substring

Returns the substring of string_exp that starts at position integer_exp1 for integer_exp2 characters or to the end of string_exp if integer_exp2 is -1. The first character in string_exp is at position 1.

Syntax

```
substring(string_exp , integer_exp1, integer_exp2)
```

TOCHeadingCount

Returns the table of contents heading count for a specific heading level.

Syntax

```
TOCHeadingCount(headingLevel)
```

Today

Returns current system date.

Syntax

```
Today()
```

trim

Returns a string_exp trimmed of leading and/or trailing blanks or trimmed of a character specified in match_character_exp. trim_what_exp may be: "LEADING", "TRAILING" or "BOTH" (default) . match_character_exp = empty string to trim blanks, or specifies a character to be trimmed. String_exp = the string to be trimmed.

Syntax

```
trim(trim_what_exp, match_character_exp, string_exp)
```

upper

Returns string_exp with all lowercase characters shifted to uppercase.

Syntax

```
upper(string_exp)
```

URLEncode

Returns the url encoded value of the input text.

Syntax

```
URLEncode('text')
```

Data Type Casting Functions

date2string

Returns a date as string in YYYY-MM-DD format.

Syntax

```
date2string(date_exp)
```

date2timestamp

Converts a date to a timestamp. The time part of the timestamp will be zero.

Syntax

```
date2timestamp(date_exp)
```

date2timestampTZ

Converts a date to a timestamp with time zone. The time and time zone parts of the timestamp will be zero.

Syntax

```
date2timestampTZ(date_exp)
```

DTinterval2string

Returns a date time interval as string in "DDDD HH:MM:SS.FFFFFFF" or "-DDDD HH:MM:SS.FFF" format.

Syntax

```
DTinterval2string(DTinterval_exp)
```

DTinterval2stringAsTime

Returns a date time interval as string. The days are converted to hours. Output format: "HHHH:MM:SS.FFFFFFF" or "HH:MM:SS.FFF".

Syntax

```
DTinterval2stringAsTime(DTinterval_exp)
```

int2DTinterval

Converts an integer to a date time interval. String_exp specifies what the integer_exp is representing: "ns"=nanoseconds, "s"=seconds (=default), "m"=minutes, "h"=hours, "d"=days.

Syntax

```
int2DTinterval(integer_exp, string_exp)
```

int2YMinterval

Converts an integer to a year month interval. String_exp specifies what the integer_exp is representing: "y"=years, "m"=months (=default).

Syntax

```
int2YMinterval(integer_exp, string_exp)
```

number2string

Converts a number to a string, using the %g format specifier (C/C++ syntax).

Syntax

```
number2string(num_exp)
```

string2date

Returns a date string as date in "YYYY-MM-DD" format.

Syntax

```
string2date(string_exp)
```

string2double

Returns a floating point number. The string argument to the function "string2double" has the following form:"[whitespace] [sign] [digits] [digits] [{d | D |e | E }[sign]digits]"

Syntax

```
string2double(string_exp)
```

string2DTinterval

Returns a date-time interval string as Date Time Interval in "[-]DD HH:MM[:SS[.FFF]]" format.

Syntax

```
string2DTinterval(string_exp)
```

string2int32

Returns an integer. The string argument to the function "string2int32" has the following form: "[whitespace] [{+ | -}] [digits]"

Syntax

```
string2int32(string_exp)
```

string2int64

Returns a long integer. The string argument to the function "string2int64" has the following form: "[whitespace] [{+ | -}] [digits]"

Syntax

```
string2int64(string_exp)
```

string2time

Returns a time string as time in "HH:MM:SS.FFFFFFF" format.

Syntax

```
string2time(string_exp)
```

string2timestamp

Returns a timestamp string as timestamp in "YYYY-MM-DD [white space]+ HH:MM:SS.FFFFFFF" format.

Syntax

```
string2timestamp(string_exp)
```

string2timestampTZ

Returns a timestamp with time zone "YYYY-MM-DD HH:MM:SS.FFFFFFF +HHMM" or "YYYY-MM-DD HH:MM:SS.FFF -HHMM" format.

Syntax

```
string2timestampTZ(string_exp)
```

string2YMinterval

Returns a year-month interval string as Year Month Interval in "[-]YY MM" format.

Syntax

```
string2YMinterval(string_exp)
```

time2string

Returns a time as string in HH:MM:SS.FFF format.

Syntax

```
time2string(time_exp)
```

timestamp2date

Converts a timestamp to a date. The time part of the timestamp will be ignored.

Syntax

```
timestamp2date(timestamp_exp)
```

timestamp2string

Returns a timestamp as string in YYYY-MM-DD HH:MM:SS.FFFFFFF format.

Syntax

```
timestamp2string(timestamp_exp)
```

timestamp2timestampTZ

Converts a timestamp to a timestamp with time zone. The displacement part of the timestamp with time zone will be zero.

Syntax

```
timestamp2timestampTZ(timestamp_exp)
```

timestampTZ2date

Converts a timestamp with time zone to a date. The time and time zone parts of the timestamp will be ignored.

Syntax

```
timestampTZ2date(timestampTZ_exp)
```

timestampTZ2string

Returns a timestamp with time zone as string in "YYYY-MM-DD HH:MM:SS.FFFFFFF +HHMM" or "YYYY-MM-DD HH:MM:SS.FFF -HHMM" format.

Syntax

```
timestampTZ2string(timestampTZ_exp)
```

timestampTZ2timestamp

Converts a timestamp with time zone to a timestamp. The displacement part of the timestamp with time zone will be ignored.

Syntax

```
timestampTZ2timestamp(timestampTZ_exp)
```

timeTZ2string

Returns a time with time zone as string in "HH:MM:SS.FFF +HHMM" or "HH:MM:SS.FFFFFFF -HHMM" format.

Syntax

```
timeTZ2string(timeTZ_exp)
```

Example

```
"-05:30" means a TimeZone of GMT minus 5 hours and 30 minutes.
```

Result: `GMT minus 5 hours and 30 minutes`

YMinterval2string

Returns a year month interval as string in "(YY MM)" or "-(YY MM)" format.

Syntax

```
YMinterval2string(YMinterval_exp)
```

Appendix C: Data Formatting Reference

This chapter contains definitions of data formatting properties found in Framework Manager. The definition for each formatting property is also shown when you select a property in the **Data Format** dialog box in Framework Manager.

Note that for SAP BW metadata:

- any unit of meaure information that exists in SAP BW is automatically appended to the data value
- you cannot define a format for each currency in a multi-currency query subject.

Data Formatting Properties

The following is a list of properties available in the data formatting dialog.

"Not Applicable" Characters

Specifies the characters to be displayed when the value to be formatted was not applicable. The default value is two dashes (--). Note that the format will be applied only if the data source supports this error condition.

Any Error Characters

Specifies the characters to be displayed when the value to be formatted was not available because of an error. This property is overridden by the more specific formatting error conditions, such as Security Error Characters. The default value is two dashes (--). Note that the format will be applied only if the data source supports this error condition.

Calendar Type

Specifies the type of calendar to be displayed. The date values will be mapped to the selected calendar before being formatted. The default value is inherited from the user's content language. Note that the Japanese Imperial setting is only applicable for Japanese languages.

Clock

Specifies whether to display the time in 12-hour or 24-hour format. The default value is inherited from the user's content language.

Currency

Specifies the currency to be used. The default currency symbol will be displayed unless the values of the Currency Display and Currency Symbol properties are changed. The default value is inherited from the model.

Currency Display

Specifies whether to display the international or local currency symbol. By default, the local currency symbol is displayed.

Currency Symbol

Specifies a character or characters to use as the symbol to identify the local currency. This symbol will precede the number and any sign, even if it is a leading sign. A space between the symbol and the numeric value can be specified by entering it in this property, after the symbol. The default value is inherited from the user's content language.

Currency Symbol Position

Specifies where the currency symbol will appear. If End is selected, any spaces that follow the character or characters in the Currency Symbol or International Currency Symbol properties will be rendered between the number and the symbol. The default value is inherited from the user's content language.

Date Ordering

Specifies the order in which to display the day, month, and year. The default value is inherited from the user's content language.

Date Separator

Specifies the character to be displayed between the year, month, and day. The default value is inherited from the user's content language.

Date Style

Specifies the date style. The results rendered are determined by the language. Generally, Short uses only numbers, Medium uses some abbreviated words, Long uses complete words, and Full includes all available details.

Decimal Separator

Specifies the character that will separate non-decimal numbers from decimals. This property is ignored if no decimals are displayed. The default value is inherited from the user's content language.

Display AM / PM Symbols

Specifies whether to display the AM or PM symbols. The default value is inherited from the user's content language.

Display As Exponent

Specifies whether to render values in scientific notation, using exponents. If this property is set to No, scientific notation will not be used. If this property is not specified, scientific notation will be used only when values exceed the maximum number of digits. The default value is inherited from the user's content language.

Display Days

Specifies whether to display the day. The format of the day can be controlled by selecting one of the specific formats. Selecting Julian means that the 3-digit day of the year will be displayed. The default value is inherited from the user's content language.

Display Eras

Specifies whether to display the era. The default value is inherited from the user's content language.

Display Hours

Specifies whether to display the hours. The default value is inherited from the user's content language.

Display Milliseconds

Specifies whether to display the milliseconds. The format of the milliseconds can be controlled by selecting one of the specific formats. This property is ignored if seconds are not displayed. The default value is inherited from the user's content language.

Display Minutes

Specifies whether to display the minutes. The format of the minutes can be controlled by selecting one of the specific formats. The default value is inherited from the user's content language.

Display Months

Specifies whether to display the month. The format of the month can be controlled by selecting one of the specific formats. The default value is inherited from the user's content language.

Display Months

Specifies whether to display the month.

Display Seconds

Specifies whether to display the seconds. The format of the seconds can be controlled by selecting one of the specific formats. The default value is inherited from the user's content language.

Display Time Zone

Specifies whether to display the time zone. The default value is inherited from the user's content language.

Display Weekdays

Specifies whether to display the weekday. The format of the weekday can be controlled by selecting one of the specific formats. The default value is inherited from the user's content language.

Display Years

Specifies whether to display the year. The first two digits of the year, which indicate the century, can be controlled by selecting one of the associated property values. The default value is inherited from the user's content language.

Display Years

Specifies whether to display the year.

Divide By Zero Characters

Specifies the characters to be displayed when a numeric value is the result of a division by zero. The default value is /0. Note that the format will be applied only if the data source supports this error condition.

Exponent Symbol

Specifies the character to be displayed to identify exponents if the scientific notation is used. The symbol will be rendered after the number, separated by a space. The default value is inherited from the user's content language.

Group Size (digits)

Specifies the primary grouping size. If a value is specified it represents the number of digits to the left of the decimal point to be grouped together and separated by the thousands separator. The default value is inherited from the user's content language.

International Currency Symbol

Specifies a character or characters to use as a symbol to identify the international currency. This symbol will replace the currency symbol. A space between the symbol and the numeric value can

be specified by entering it in this property, after the symbol. The default value is inherited from the user's content language.

Mantissa (digits)

Specifies the number of digits to be displayed following the exponent symbol if the scientific notation is used.

Maximum No. of Digits

Specifies the maximum number of digits that can be displayed. If the maximum number of digits is not sufficient to display the value, a scientific notation will be used. The default value is inherited from the user's content language.

Minimum No. of Digits

Specifies the minimum number of digits that can be displayed. If the minimum number of digits is too high to display a value, the padding character will be used. The default value is inherited from the user's content language.

Missing Value Characters

Specifies the character or characters to be displayed when the value is missing. If no value is entered for this property, an empty string will be displayed.

Negative Pattern

Specifies a presentation format, based on patterns, for negative numbers. Some restrictions exist. The numerical part of the negative pattern is ignored. Only the suffix and the prefix are used. For example, in the pattern ABC#,##0.#EFG, ABC is the prefix, EFG is the suffix and #,##0.# is the numerical part of the pattern.

Negative Sign Position

Specifies where the negative sign will appear. The default value is inherited from the user's content language.

Negative Sign Symbol

Specifies how to display negative numbers. The default value is inherited from the user's content language.

No. of Decimal Places

Specifies the number of digits to be displayed to the right of the decimal point. If this property is not set, the number of decimal places will vary depending on the number rendered.

Numeric Overflow Characters

Specifies the characters to be displayed when a numeric value is the result of a numeric overflow. The default value is two dashes (--). Note that the format will be applied only if the data source supports this error condition.

Padding Character

Specifies the character that will be used to pad values that have fewer digits than the minimum number of digits. The default value is inherited from the user's content language.

Pattern

Specifies a presentation format that is based on patterns.

Percentage Symbol

Specifies whether to display the values per hundred (percent) or per thousand. The symbol will be appended to the number and any trailing sign. A space between the numeric value and the symbol can be specified by entering it in this property, after the symbol. The default value is inherited from the user's content language.

Percent Scale (integer)

Scale to be applied to value after formatting. If omitted, no percent scale will be applied and the value will formatted according the normal decimal positioning associated with the percent (or per mille) symbol.

Scale

Specifies how many digits to move the decimal delimiter for formatting purposes. For example, move the decimal three spaces to present values in thousands. The default value is inherited from the database field.

Secondary Group Size (digits)

Specifies the secondary grouping size. If a value is specified it represents the number of digits to the left of the primary group that will be grouped together and separated by the thousands separator. If this property is left blank, the secondary grouping of digits is the same number as the primary group size, as specified by the Group Size (digits) property. The default value is inherited from the user's content language.

Security Error Characters

Specifies the characters to be displayed when the value to be formatted was not available for security reasons. The default value is #!Security. Note that the format will be applied only if the data source supports this error condition.

Thousands Separator

Specifies how to delimit digit groups, such as thousands. This property is only used if the Use Thousands Separator property is set to Yes. The default value is inherited from the user's content language.

Time Separator

Specifies the character to be displayed between the hour, minute, and second. The default value is inherited from the user's content language.

Time Style

Specifies the time style to be displayed. The exact results that will be rendered are determined by the language. Generally, Short means that the minimum details will be displayed, Long adds seconds, and Full means that all details are displayed, including the time zone. The default value is inherited from the user's content language.

Time Unit

Specifies the unit of measure of the value. This property will be ignored if any day or time components are shown. The default value is inherited from the user's content language.

Use Thousands Separator

Specifies whether the grouping delimiter will be applied as defined by the Group Size property. The default value is inherited from the user's content language.

Zero Value Characters

Specifies the character or characters to be displayed when the value is zero (0). If no value is entered for this property, the Maximum No. of Digits property determines how many zero digits are displayed.

Appendix D: Using Patterns to Format Data

You can format data so that it matches any pattern of text and numbers when default formats are not appropriate. For example, you can format dates to use full text including the era, or you can format them to only use numbers and show the last two digits of years to save space.

Using symbols and patterns can provide similar results as basic data formatting tasks. For example, you can set how many digits appear after the decimal point. You can achieve these types of results with a pattern, or you can set the No. of Decimal Places property. Patterns allow flexibility for more complex requirements.

Each supported content language code requires a specific set of symbols to be used in patterns. For each language code, there are two tables you will need; one for date and time symbols, and one for decimal symbols. The decimal symbols are the same for all locales, however, date and time symbols are grouped into six locale groups. Check the Date and Time Symbol section to see which locale group is used for your locale.

To define patterns, open the **Data Format** dialog box, and edit the **Pattern** property for each format type. Use the symbols that are defined in the language code tables, and follow these guidelines.

Pattern Guidelines

When you define a pattern, the number of symbols you use affects how the data will be shown. There are different rules for text, numbers, and values that can take the form of text or numbers.

Text

You can specify whether text is produced in full or abbreviated form.

Number of symbols	Meaning	Example
4 or more	Full text form	EEEE produces Monday
Less than 4	Abbreviated form	EEE produces Mon

Numbers

The number of symbols you use in a pattern sets the minimum number of digits that are produced in a report. Numbers that have fewer digits than specified are zero-padded. For example, if you specify mm for minutes, and the database value is 6, the report will show 06.

Note: The year value is handled differently. If you specify two symbols for year, the last two digits of the year value is produced. For example, yyyy produces 1997, and yy produces 97.

Text and Numbers

For values that can produce text or numbers, such as months, you can specify whether text or numbers are produced, and whether words are abbreviated.

Number of symbols	Meaning	Example
3 or more	Text	MMMM produces January MMM produces Jan
Less than 3	Numbers	MM produces 01 M produces 1

Date and Time Symbols

Locale ID	Locale group
af-za, en, en-au, en-be, en-bw, en-ca, en-gb, en-hk, en-ie, en-in, en-mt, en-nz, en-ph, en-sg, en-us, en-vi, en-za, fo-fo, gl-es, id, id-id, is, is-is, it, it-ch, it-it, kk-kz, ms, ms-bn, ms-my, nb-no, nl, nl-be, nl-nl, no, no-no, om-et, om-so, pl, pl-pl, pt, pt-br, pt-pt, so-dj, so-et, so-ke, so-so, sv, sv-fi, sv-se, sw-ke, sw-tz	Locale Group A

Locale ID	Locale group
be-by, bg-bg, el, el-gr, fi, fi-fi, hr, hr-hr, hu, hu-hu, ja, ja-jp, ko, ko-kr, ro, ro-ro, ru, ru-ua, ru-ru, sh-yu, sk, sk-sk, sl-si, sq-al, sr-sp, th, tr, tr-tr, uk-ua, zh, zh-cn, zh-hk, zh-mo, zh-sg, zh-tw	Locale Group B
ca-es, cs, cs-cz, da, da-dk, es, es-ar, es-bo, es-cl, es-co, es-cr, es-do, es-ec, es-es, es-gt, es-hn, es-mx, es-ni, es-pa, es-pe, es-pr, es-py, es-sv, es-us, es-uy, es-ve, eu-es, mk-mk	Locale Group C
de, de-at, de-be, de-ch, de-de, de-lu	Locale Group D
fr, fr-be, fr-ca, fr-ch, fr-fr, fr-lu	Locale Group E
ga-ie	Locale Group F

Locale Group A

Locales: af-za, en, en-au, en-be, en-bw, en-ca, en-gb, en-hk, en-ie, en-in, en-mt, en-nz, en-ph, en-sg, en-us, en-vi, en-za, fo-fo, gl-es, id, id-id, is, is-is, it, it-ch, it-it, kk-kz, ms, ms-bn, ms-my, nb-no, nl, nl-be, nl-nl, no, no-no, om-et, om-so, pl, pl-pl, pt, pt-br, pt-pt, so-dj, so-et, so-ke, so-so, sv, sv-fi, sv-se, sw-ke, sw-tz

Meaning	Symbol	Presentation	Example
Era	G	Text	AD
Year	y	Number	1996
Year (of 'Week of Year')	Y	Number	1996

Meaning	Symbol	Presentation	Example
Month in year	M	Text and number	July and 07
Week in year	w	Number	27
Week in month	W	Number	2
Day in year	D	Number	189
Day in month	d	Number	10
Day of week in month	F	Number	2 (2nd Wed in July)
Day of Week (1=first day)	e	Number	2
Day in week	E	Text	Tuesday
a.m. or p.m. marker	a	Text	pm
Hour in day (1 to 24)	k	Number	24
Hour in a.m. or p.m. (0 to 11)	K	Number	0
Hour in a.m. or p.m. (1 to 12)	h	Number	12
Hour in day (0 to 23)	H	Number	0
Minute in hour	m	Number	30
Second in minute	s	Number	55
Millisecond	S	Number	978
Time zone	z	Text	Pacific Standard Time
Escape used in text	'	n/a	n/a
Single quote	''	n/a	'

Locale Group B

Locales: be-by, bg-bg, el, el-gr, fi, fi-fi, hr, hr-hr, hu, hu-hu, ja, ja-jp, ko, ko-kr, ro, ro-ro, ru, ru-ua, ru-ru, sh-yu, sk, sk-sk, sl-si, sq-al, sr-sp, th, tr, tr-tr, uk-ua, zh, zh-cn, zh-hk, zh-mo, zh-sg, zh-tw

Meaning	Symbol	Presentation	Example
Era	G	Text	AD
Year	a	Number	1996
Year (of 'Week of Year')	A	Number	1996
Month in year	n	Text and number	July and 07
Week in year	w	Number	27
Week in month	W	Number	2
Day in year	D	Number	189
Day in month	j	Number	10
Day of week in month	F	Number	2 (2nd Wed in July)
Day of Week (1=first day)	e	Number	2
Day in week	E	Text	Tuesday
a.m. or p.m. marker	x	Text	pm
Hour in day (1 to 24)	h	Number	24
Hour in a.m. or p.m. (0 to 11)	K	Number	0
Hour in a.m. or p.m. (1 to 12)	k	Number	12
Hour in day (0 to 23)	H	Number	0
Minute in hour	m	Number	30
Second in minute	s	Number	55

Meaning	Symbol	Presentation	Example
Millisecond	S	Number	978
Time zone	z	Text	Pacific Standard Time
Escape used in text	'	n/a	n/a
Single quote	''	n/a	'

Locale Group C

Locales: ca-es, cs, cs-cz, da, da-dk, es, es-ar, es-bo, es-cl, es-co, es-cr, es-do, es-ec, es-es, es-gt, es-hn, es-mx, es-ni, es-pa, es-pe, es-pr, es-py, es-sv, es-us, es-uy, es-ve, eu-es, mk-mk

Meaning	Symbol	Presentation	Example
Era	G	Text	AD
Year	u	Number	1996
Year (of 'Week of Year')	U	Number	1996
Month in year	M	Text and number	July and 07
Week in year	w	Number	27
Week in month	W	Number	2
Day in year	D	Number	189
Day in month	t	Number	10
Day of week in month	F	Number	2 (2nd Wed in July)
Day of Week (1=first day)	e	Number	2
Day in week	E	Text	Tuesday
a.m. or p.m. marker	a	Text	pm
Hour in day (1 to 24)	h	Number	24

Meaning	Symbol	Presentation	Example
Hour in a.m. or p.m. (0 to 11)	K	Number	0
Hour in a.m. or p.m. (1 to 12)	k	Number	12
Hour in day (0 to 23)	H	Number	0
Minute in hour	m	Number	30
Second in minute	s	Number	55
Millisecond	S	Number	978
Time zone	z	Text	Pacific Standard Time
Escape used in text	'	n/a	n/a
Single quote	''	n/a	'

Locale Group D

Locales: de, de-at, de-be, de-ch, de-de, de-lu

Meaning	Symbol	Presentation	Example
Era	G	Text	AD
Year	j	Number	1996
Year (of 'Week of Year')	J	Number	1996
Month in year	M	Text and number	July and 07
Week in year	w	Number	27
Week in month	W	Number	2
Day in year	D	Number	189
Day in month	t	Number	10
Day of week in month	F	Number	2 (2nd Wed in July)

Meaning	Symbol	Presentation	Example
Day of Week (1=first day)	e	Number	2
Day in week	E	Text	Tuesday
a.m. or p.m. marker	a	Text	pm
Hour in day (1 to 24)	h	Number	24
Hour in a.m. or p.m. (0 to 11)	K	Number	0
Hour in a.m. or p.m. (1 to 12)	k	Number	12
Hour in day (0 to 23)	H	Number	0
Minute in hour	m	Number	30
Second in minute	s	Number	55
Millisecond	S	Number	978
Time zone	z	Text	Pacific Standard Time
Escape used in text	'	n/a	n/a
Single quote	''	n/a	'

Locale Group E

Locales: fr, fr-be, fr-ca, fr-ch, fr-fr, fr-lu

Meaning	Symbol	Presentation	Example
Era	G	Text	AD
Year	a	Number	1996
Year (of 'Week of Year')	A	Number	1996
Month in year	M	Text and number	July and 07

Meaning	Symbol	Presentation	Example
Week in year	w	Number	27
Week in month	W	Number	2
Day in year	D	Number	189
Day in month	j	Number	10
Day of week in month	F	Number	2 (2nd Wed in July)
Day of Week (1=first day)	e	Number	2
Day in week	E	Text	Tuesday
a.m. or p.m. marker	x	Text	pm
Hour in day (1 to 24)	h	Number	24
Hour in a.m. or p.m. (0 to 11)	K	Number	0
Hour in a.m. or p.m. (1 to 12)	k	Number	12
Hour in day (0 to 23)	H	Number	0
Minute in hour	m	Number	30
Second in minute	s	Number	55
Millisecond	S	Number	978
Time zone	z	Text	Pacific Standard Time
Escape used in text	'	n/a	n/a
Single quote	''	n/a	'

Locale Group F

Locales: ga-ie

Meaning	Symbol	Presentation	Example
Era	R	Text	AD
Year	b	Number	1996
Year (of 'Week of Year')	B	Number	1996
Month in year	M	Text and number	July and 07
Week in year	t	Number	27
Week in month	T	Number	2
Day in year	l	Number	189
Day in month	L	Number	10
Day of week in month	F	Number	2 (2nd Wed in July)
Day of Week (1=first day)	e	Number	2
Day in week	E	Text	Tuesday
a.m. or p.m. marker	a	Text	pm
Hour in day (1 to 24)	u	Number	24
Hour in a.m. or p.m. (0 to 11)	K	Number	0
Hour in a.m. or p.m. (1 to 12)	k	Number	12
Hour in day (0 to 23)	U	Number	0
Minute in hour	n	Number	30
Second in minute	s	Number	55
Millisecond	S	Number	978
Time zone	c	Text	Pacific Standard Time

Meaning	Symbol	Presentation	Example
Escape used in text	'	n/a	n/a
Single quote	''	n/a	'

Decimal Format Symbols

All locales

Symbol	Meaning
0	A digit that is shown even if the value is zero.
#	A digit that is suppressed if the value is zero.
.	A placeholder for decimal separator.
,	A placeholder for thousands grouping separator.
E	Separates mantissa and exponent for exponential formats.
;	Separates formats for positive numbers and formats for negative numbers.
-	The default negative prefix.
%	Multiplied by 100, as percentage.
‰	Multiplied by 1000, as per mille.
¤	The currency symbol. If this symbol is present in a pattern, the monetary decimal separator is used instead of the decimal separator.
¤¤	The international currency sign. It will be replaced by an international currency symbol. If it is present in a pattern, the monetary decimal separator is used instead of the decimal separator.
X	Other characters that can be used in the prefix or suffix.
'	Used to quote special characters in a prefix or suffix.
/u221E	Infinity symbol.
/uFFFD	Not a Number symbol.

Appendix E: Guidelines for Working with SAP BW Data in Other Applications

You can use data and metadata in a model in other applications. This functionality applies to all supported data sources, including SAP BW. Special considerations must be given when using SAP BW metadata and data in other applications.

Working with SAP BW Data in Transformer 8.x and 7.x

The content of this appendix applies to both Transformer 8.x and Transformer 7.x.

Transformer 8.3, can leverage metadata from other published OLAP packages including SAP BW packages. As a result, Transformer PowerCubes can be used as high speed data access cache methods for distributing smaller or focused areas of your business information. You add dimensions from OLAP packages directly from the dimension map. To add fact information from an SAP package, create a Cognos 8 report that filters the OLAP source to the required data segment and then export that data to a CSV file that can be used as the Transformer model fact query.

In Transformer 7.x, the only way to use SAP BW data is through externalized CSV files.

When publishing a Framework Manager package, you can externalize query subjects and dimensions into formats that you can use in Transformer 7.x or other applications. This functionality applies to all supported data sources. Special considerations must be given when externalizing models based on SAP BW metadata.

When extracting data from SAP BW using Framework Manager, you must understand the distinction that Framework Manager makes between different types of dimensions. Each type of dimension exhibits a different behavior when it is externalized, and can be modified before externalizing.

Note: Externalized queries will not be supported in future releases of C8 Business Intelligence Transformer.

Extract Size

The **Extract Size** data source property within Framework Manager is used to control the amount of data retrieved from SAP BW at any one time.

If this setting is negative, zero, or empty, a single query is issued to SAP BW to extract the characteristic data.

If this setting is a positive value, Framework Manager issues multiple queries to SAP BW, each of which returns approximately the number of megabytes specified by the **Extract Size** property.

This feature can reduce the overall size of the query result on the SAP BW server. Overall query execution may take longer, but for large characteristics, not using this feature may result in consumption of a user's allotted memory space on the SAP BW server.

The entire data for a characteristic dimension will be in memory within Framework Manager prior to the production of an extract file. It is important that only the required query items be extracted from SAP BW to ensure that an extract does not fail due to memory allocation errors within Framework Manager.

Model query subjects are extracted using the same mechanism by which queries are executed within Cognos 8. Therefore, the **Extract Size** property has no effect on the query execution.

Measure Dimensions

When extracting a measure dimension, you should create a model query subject containing the measures that you want. You should include the business key query item from each of the levels of each dimension, depending on the level of granularity that you are trying to achieve.

For information about externalizing model query subjects, see "Framework Manager Considerations" (p. 560).

Characteristic Dimensions

Characteristic dimensions are externalized independent of the type of SAP BW data source, such as InfoCube or SAP BW Query.

Framework Manager uses a single approach to externalize all dimensions that do not contain fact query items. In these cases, the extract size configuration setting is used to control the manner in which data is extracted from SAP BW.

Note: Model query subjects are externalized in a different manner, regardless of whether they contain fact query items or not. For information about externalizing model query subjects, see "Framework Manager Considerations" (p. 560).

Key Figures Dimensions from an SAP BW InfoCube

When externalizing the key figures dimension from a model based on an InfoCube, Framework Manager uses exactly the same approach as used for externalizing model query subjects.

For an InfoCube containing more than a few thousand transactions, externalizing an InfoCube directly from Framework Manager can easily exceed both time and memory limits on either the client or server. In such cases, it is highly recommended that an SAP BW Query be used as the basis for externalizing the SAP BW metadata.

Key Figures Dimensions from an SAP BW Query

Using a BEx query as the basis for externalizing key figures from an SAP BW data source is, in most cases, the best approach. By using a BEx variable to break the data of the key figure dimension into manageable sections, arbitrarily large volumes of transaction data can be extracted from SAP BW.

Note, however, that this approach incurs some restrictions as to what can be extracted from SAP BW, and how it can be extracted. The remainder of this section describes how an SAP BW Query is used to extract data from SAP BW, including all known restrictions and limitations.

SAP BW Query Requirements

For the remainder of this section, it is assumed that an SAP BW Query is being used as the basis for externalizing the data, not as the basis for reporting, and with the intent of exceeding the memory and time limitations associated with extracting data directly from an InfoCube.

It is not possible to externalize an arbitrary SAP BW Query. An SAP BW Query must adhere to the following restrictions if you want to externalize it:

- Only a single field can be shown. Showing something such as "key and text" will result in incorrect data.

  To change what appears for a characteristic, right-click the characteristic along the edge of the SAP BW Query and click **Properties**. In the **Properties** dialog box, set the **Display As** value to Key.

  We strongly recommend that you use Key.

- To reduce data volumes, as well as the amount of aggregation performed by the SAP BW server, we strongly recommend that summarization for all characteristics in the query be disabled in its property sheet.

  To disable summarization for a characteristic, right-click the characteristic along the edge of the SAP BW Query and click **Properties**. In the **Properties** dialog box, set the **Suppress Results Rows** value to Always.

- If at least one characteristic in an InfoQuery is displayed as something other than Key, then summarization for all characteristics must be suppressed.

- The query must not contain the Currency/Unit characteristic.

- None of the characteristics may be assigned a display hierarchy, either explicitly or by a variable.

- If a characteristic is included in an SAP BW Query as a free characteristic, no values will appear for that characteristic in the key figures extract.

  A filter on a free characteristic acts as a filter on the data returned by SAP BW. It is an efficient mechanism for defining a subset of an InfoCube.

  Such a filter may also be applied to a characteristic along an axis of an SAP BW Query, in which case the filtered values appear in the key figures extract.

- All key figures in the SAP BW Query must be numeric.

- The values of each key figure should be in a single currency since currency information is not carried forward into a CSV extract. A variable should not be used to drive the assignment of a target currency.

- Include in the SAP BW Query only those characteristics which are to be extracted using Framework Manager. Including unnecessary characteristics increases the volume of data transferred from SAP BW, thus affecting performance.

Guidelines and Constraints When Working with SAP BW Cubes

You must use CSV files when importing metadata from SAP BW cubes. For performance reasons, we recommend that you filter on geography, time periods, or some other dimension that limits the amount of data retrieved. Remember to apply your dimension filter to the related dimensions and their fact tables (measures). For more information, see "Create a Filter" (p. 233).

Because SAP BW cubes are multidimensional, rollups are applied at the source. If you change the rollup type after importing the data into Transformer, your results will not be valid.

Missing data or metadata that is out-of-scope for a particular measure may yield different results, depending on the context. You may see:

- NULL values
- # symbols
- REST_H
- Not assigned

Because such duplicate tokens can cause problems in Transformer, in unique levels for example, we recommend that you assign filters to the dimension so that they do not appear in the imported data.

Finally, remember to select only those query items needed to generate your filtered data.

Framework Manager Considerations

When extracting the measure dimension from an SAP BW Query, the **Extract Size** property of the data source is used to control the amount of data retrieved from the SAP BW server at one time. Model query subjects are externalized in a different manner, regardless of whether they contain fact query items or not. In this scenario, the setting has no affect on the SAP BW server, but it does limit the amount of memory allocated at any one time by Framework Manager to retrieve the data.

It is important to realize that filters defined on the key figures dimension are not enforced when extracting data from an SAP BW Query. To obtain performance benefits of extracting data from an SAP BW Query, filters must be defined in an SAP BW Query.

In addition, any calculations defined within the key figures dimension are ignored. These may be defined either within the SAP BW Query in BEx, or in a model query subject in Framework Manager.

Each characteristic extracted must contain at least one query item from the lowest level of its hierarchy (if there is one) to provide linkage with the key figures extract. You should include the business key query item from each of the levels of each dimension, depending on the level of granularity that you are trying to achieve.

Use of Variables to Externalize Key Figures from SAP BW Query

The volume of transactions within an SAP BW Query is such that, in most cases, the use of a single query to extract the data from SAP BW will exceed the memory allocated to a user on an SAP BW server. In Framework Manager, you can use a single optional variable to extract fact data from an SAP BW Query in reasonably sized sections.

To use this feature, one characteristic included in the SAP BW Query (but not included as a free characteristic) is assigned a variable that conforms to the following restrictions:

- It must be a single value.
- It must be optional.
- It must not have a default value.
- It can be defined on the characteristic or a presentation hierarchy.

If an SAP BW Query contains such a variable and the key figures dimension is externalized, Framework Manager runs a query for each possible value associated with a variable. Thus, by choosing an appropriate characteristic, the key figures dimension can be extracted without exceeding the memory restrictions of either the client or server. Memory caches on the client and server are flushed after each query.

If a presentation hierarchy is used to drive the creation of extract sections, it is important that the values for a variable be obtained from a single level in the hierarchy, otherwise the extract will contain data summarized at different levels. To restrict the values for a variable to a single level of a hierarchy, edit the Level Restriction of the variable in Framework Manager. For example, using a value such as "2:2" indicates that only values from the second level of the hierarchy are to be used (level 0 is the root of a hierarchy).

In the presence of an SAP BW Query with one such variable, the value of the variable is reset after each query.

If an SAP BW Query contains anything more than a single variable, or one that is defined differently than described above, Framework Manager does not attempt to use a variable to break the extraction of the key figures dimension into smaller sections.

Workaround for Problems Encountered While Externalizing

When externalizing a data source from Framework Manager, you may encounter an authentication error if

- the model is published to Content Manager
- externalizing the data takes longer to perform than the timeout period assigned to passports within Cognos Configuration

Users are not prompted to re-enter their authentication credentials.

If an error occurs, the externalized data is still complete and valid. However, if the modeler chooses to actually publish the model, the modeler must re-authenticate and re-publish the model, but without externalizing the data.

Another solution is to publish the model to the network, in which case the authentication error does not occur.

Building PowerCubes from SAP BW Data

You can build Cognos PowerCubes from SAP BW data. There are guidelines to consider for both Framework Manager and Transformer.

Framework Manager Guidelines

When externalizing data for the purpose of creating one or more PowerCubes, keep these considerations in mind.

- The extract of each characteristic must have a common key query item that is equivalent to a surrogate key query item in the key figures extract.
- For an extract based on an SAP BW Query, it is strongly recommended that all characteristics be displayed as Key in the SAP BW Query.
- If a characteristic does not have a presentation hierarchy, or a new one is desired, extract one or more query items that can form the basis for levels in a hierarchy.
- During the import of SAP BW metadata into a model that is to be used for the purpose of extracting data, it is desirable to limit the model to only those query items that are absolutely required to build a PowerCube. This will have the effect of improving data extract performance.
- A practical limit for PowerCubes is 2,000,000 categories (values) for a dimension (characteristic) .

Transformer Guidelines

When using the SAP BW data that you extracted from Framework Manager, keep these considerations in mind.

- In Transformer 8.3, you can insert regular dimensions from SAP data sources directly from a Cognos 8 data source, using the **Insert dimension from package** option. For more information, see the Cognos 8 Transformer *User Guide*..
- Using the model wizard in Transformer, insert a data source of type Delimited-Field Text With Column Titles and start by selecting the CSV file. Do not run auto-design.
- Drag all the key figure columns from the Data Sources pane into the Measures pane. Ensure that the aggregation rule for each measure is correctly defined within Transformer to align as closely as possible with the aggregation rule defined in SAP BW.
- It is recommended that the storage type for all measures be set to 64-bit floating point.
- Using the date wizard, select a query item that will provide the dates for the PowerCube. Note that the dates for the PowerCube can be derived entirely from the transaction data.
- Insert the various CSV files corresponding to the characteristics that were externalized using Framework Manager.

  Each CSV file contains a column that corresponds to a column in the key figures CSV file. By right-clicking the various columns and editing the column properties, ensure the columns that provide the linkage between a characteristic and the key figures have the same name. For example, if a key figure column is named Customer and the corresponding column in the customer CSV file is named Customer - Key, then the name of the column in the key figures CSV file can be changed to Customer - Key.

- For each characteristic, create a new dimension, using the key columns, or other attributes of a characteristic, to drive the levels of the dimension. For each level, ensure that the properties for the label, short name, and description are assigned source columns, if applicable.
- For the root level of each characteristic (dimension), ensure it is marked as unique.
- SAP BW presentation hierarchies may contain ragged paths, typically in association with the "not assigned" and "#" nodes in the hierarchy. The gaps in these hierarchies produce blanks at the associated level in the Transformer hierarchy.

In Transformer, it is possible to define the text that should be used for blanks (the default text is "<blank>"). A best practice is to define a more appropriate text for blank entries for all such levels.

Work with SAP BW Data in Cognos 8 Planning

For information about detailed fact query subjects that are used when working with SAP BW Data in Cognos 8 Planning, see the Cognos 8 Planning Analyst *User Guide*.

Appendix F: Reserved Words

You must ensure that the names of data sources, tables, and columns do not use names reserved by Cognos 8.

If you must use a reserved word, enclose the word in quotes in the SQL specification. For example, `select Orderdate, "Timezone"`.

Cognos 8 reserves these words:

_cursor	in	row
_local	inner	row_number
_rowset	insert	rows
abs	int	rsum
all	integer	scroll
and	intersect	second
any	interval	select
as	into	set
asc	is	sets
at	join	smallint
avg	last	snapshot
between	last_value	some
bigint	leading	sqrt
bit_length	left	stddev
boolean	like	stddev_pop
both	ln	stddev_samp
by	local	substring

call	localtime	sum
case	localtimestamp	table
cast	lower	tertile
ceil	max	then
ceiling	min	time
char	minute	timestamp
char_length	mod	timezone_hour
character	month	timezone_minute
character_length	national	to
coalesce	natural	trailing
count	nchar	trim
create	no	true
cross	not	unbounded
cube	ntile	union
cume_dist	null	unknown
current	nullif	updatable
current_date	nulls	update
current_time	numeric	upper
current_timestamp	nvarchar	user
cursor	octet_length	using
date	of	values
day	on	var_pop
dbkey	or	var_samp

dec	order	varchar
decimal	others	variance
declare	out	varying
delete	outer	when
dense_rank	over	where
desc	partition	window
distinct	perc	with
double	percent_rank	within
else	percentile	without
end	percentile_cont	xavg
escape	percentile_disc	xcount
except	position	xfirst
exists	power	xlast
exp	preceding	xmax
extract	precision	xmin
false	prefilter	xmovingavg
filename	qualify	xmovingsum
filter	range	xntile
first	rank	xperc
first_value	ratio_to_report	xrank
float	ravg	xratio
floor	rcount	xstddev
following	rdiff	xstddev_pop

for	real	xsum
from	recursive	xtertile
full	returning	xvariance
group	right	xvariance_pop
grouping	rmax	year
having	rmin	zone
hour	rollup	

Appendix G: XML Data Types

You can import XML as a tabular data source in Framework Manager.

The following data types are supported when importing from XML into Framework Manager.

boolean	int	NOTATION
byte	integer	positiveInteger
date	language	QName
dateTime	long	short
decimal	Name	string
double	NCName	time
ENTITIES	negativeInteger	token
ENTITY	NMTOKEN	unsignedLong
float	NMTOKENS	unsignedInt
ID	nonNegativeInteger	unsignedShort
IDREF	NonPositiveInteger	unsignedByte

The following data types are not supported.

base64Binary	gMonthDay
duration	gMonth
gYearMonth	hexBinary
gYear	

Glossary

access permissions

Rules defining the access rights to resources. Access permissions can be granted to any combination of namespaces, groups, or users. Examples of resources are reports and folders.

alias

In modeling and database terminology, a secondary name for a database table. Aliases are used to create a distinct reference to the table in the model, so that self-joins can be created or ambiguous query paths can be resolved.

In map information technology, a secondary name for a map feature. Aliases are used to create a reference between custom map feature names and feature names in databases.

attribute

In relational models, a query item that is not a measure or identifier. When a query item is an attribute, it is not intended to be aggregated, or used for grouping or generating prompt pick lists.

In dimensional models, attributes provide qualitative information about members of a level in a dimension. For example, the Store level within the Retailer dimension might have properties such as "address" or "retail space." In general, dimensional attributes do not have measure values or rollups associated with them, but are used to locate or filter members.

calculated member

A member of a dimension whose measure values are not stored but are calculated at run time using an expression.

cardinality

For OLAP data sources, cardinality is the number of members in a hierarchy. The cardinality property for a hierarchy is used to assign solve orders to expressions. For relational data sources, cardinality indicates the nature of the relationship between two query subjects, query items, or other model objects.

conformed dimension

A dimension, with a single definition, that is reused or shared across multiple data subject areas. Physically, these subject areas can include multiple cubes or multiple relational star schemas. Conformed dimensions provide a useful design approach for dimensions that are widely used throughout the enterprise, for example Product or Time. This is because common definitions for common dimensions allow data from different subject areas to be meaningfully compared and used in calculations and drill through from one area to another.

cube

A physical data source containing a multidimensional representation of data. A cube contains information organized into dimensions and optimized to provide faster retrieval and navigation in reports. In Cognos Planning, a cube (see also D-Cube) corresponds to a tab on Contributor client user interface.

data source

A relational database, dimensional cube, file, or other physical data store that can be accessed though Cognos 8.

dimension

In Cognos Planning, the rows, columns, and pages of a cube are created from dimensions. Dimensions are lists of related items such as Profit and Loss items, months, products, customers, and cost centers. Dimensions also contain all the calculations. One dimension can be used by many cubes.

In Cognos 8 BI a dimension is a broad grouping of descriptive data about a major aspect of a business, such as products, dates, or markets. Each dimension includes different levels of members in one or more hierarchies and an optional set of calculated members.

governor

A set of rules to limit user activities, such as the execution of reports, that either take too long, or consume too many resources.

hierarchy

A hierarchy represents a collection of dimensional members organized into a tree structure, with each member having one or more parent members and an arbitrary number of child members.

The root of a hierarchy has no parent, and leaf members of a hierarchy have no children.

level

A level is a set of members that have common attributes. For example, a geographical dimension might contain levels for country, region, and city. Levels are used to define the structure of hierarchies in a dimension.

locale

A code that is used to set the language or dialect used for browsers, report text, and so on; and the regional preferences, such as formats for time, date, money, and money expressions.

For Cognos products, you can specify a locale for the product interface (product locale) and for the data in the report (content locale).

measure

A performance indicator that is quantifiable and used to determine how well a business is operating. For most business purposes, the aggregate values of a measure are more useful than individual

values. For example, measures can be Revenue, Revenue/Employee, and Profit Margin %. In relational modeling, this is also called a "fact."

member

A member is a unique item within a hierarchy. For example, Camping Equipment and 4 Man tent are members of the Products hierarchy.

See also member unique name.

model

A physical or business representation of the structure of the data from one or more data sources. A model describes data objects, structure, and grouping, as well as relationships and security.

In Cognos 8 BI, a design model is created and maintained in Framework Manager. The design model or a subset of the design model must be published to the Cognos 8 server as a package for users to create and run reports.

In Cognos Planning, a model is a group of D-Cubes, D-Lists, D-Links, and other objects stored in a library. A model may reside in one or more libraries, with a maximum of two for Contributor.

model segment

A part of a Framework Manager project that is a shortcut to a second project. Segments can be parameter maps, data sources, namespaces, or folders. You must make any changes in the source project, not in the segment.

You use segments to simplify model maintenance or to facilitate multi-user modeling. How you use segments does not affect the packages that you publish to Cognos 8.

namespace

For authentication and access control, a configured instance of an authentication provider. Allows access to user and group information.

In XML, a collection of names, identified by a URI reference, which are used in XML documents as element types and attribute names.

In Framework Manager, namespaces uniquely identify query items, query subjects, and so on. You import different databases into separate namespaces to avoid duplicate names.

normalization

An industry standard approach to the design of relational databases, based on primary and foreign keys, aimed at minimizing redundancy and optimizing update operations.

package

A subset of a model, which can be the whole model, to be made available to the Cognos 8 server.

For Metric Studio users, see metric package.

project

In Framework Manager, a set of models, packages, and related information for administration, and for sharing model information.

In Metric Studio, a project is a task or set of tasks undertaken by a team and monitored on a scorecard. A project tracks the dates, resources and status of the project.

In Metric Designer, a project is a group of extracts. Each extract contains the metadata that is used to populate the Metric Studio data store or to create applications.

publish

In Cognos 8 BI, refers to the creation of a package that makes metadata available to the Cognos 8 server. Information in the package is used to create reports and other content.

In Cognos Planning, refers to a function that is used to copy the data from Contributor or Analyst to a datastore, typically so it can be used for reporting purposes.

query item

A representation of a column of data in a data source. It contains a reference to a database column, a reference to another query item, or a calculation. Query items may appear in a model or in a report.

query subject

One of the types of objects inside a model. A query subject can be defined as a collection of references to items of other query subjects, or expressed as an SQL expression that represents selected query items. These query items will be retrieved from objects such as tables, synonyms, views, and so on. Query subjects contain query items.

Index

Symbols

"Not Applicable" Characters, 537

A

accessing
  data source connections, 61
  secured InfoCube, 191
access permissions
  definition, 571
action logs, 300
  running in batch mode, 302
adding
  business rules for relational metadata, 153
  business rules for SAP BW metadata, 231
  data source functions, 317
  function sets, 317
  groups, 253
  languages for relational metadata, 135
  languages to packages, 258
  metadata security, 258
  object security, 255
  roles, 253
  security, 258
  users, 253
additive, 146, 218
administrative access, 258
advisor, model, 185
aggregate rollups, 314
aggregation, 324
  rules for relational metadata, 141, 143, 144, 146
  rules for SAP BW metadata, 216, 217, 218
  types, 218
  types for relational metadata, 146
aggregation rules, 146
Aggregation Rules property
  relational metadata, 141
aggregation type
  calculated, 185
aliases
  definition, 571
  using with parameters, 161, 237
ambiguous objects, 361
ambiguous paths, 178
ambiguous relationships, 338
Analysis Studio and sparse data, 83
analyzing
  models, 185
  problems, 29
  publishing impact on packages, 292
Any Error Characters, 537
applying
  filters for relational metadata, 158
  filters for SAP BW metadata, 235
Architect XML files
  importing, 68
As View SQL, 114
  model query subjects, 115
attributes, 115, 116, 123, 124, 199, 202, 204, 374
  definition, 571
  of a dimension, 57
auditing, 268, 292
automatic aggregation types
  relational metadata, 144
  SAP BW metadata, 217, 218

B

balanced hierarchies, 120, 200, 244
  DB2 OLAP, 54
BLOB
  in stored procedures, 381
BLOBs, 310
BmtScriptPlayer
  syntax, 303
braces in expressions, 384
branches
  creating, 272
branching
  methodologies, 270

projects, 269
repositories, 289
broken reports, 265
business layer, 79
Business Objects
imported expressions, 77
business rules for relational metadata, 153
calculations, 153
filters, 156
macros, 165
parameter maps, 161
parameters, 164
prompts, 149
session parameters, 163
business rules for SAP BW metadata, 231
calculations, 231
filters, 233
business view, 79
business view for relational metadata, 175
folders, 183
namespaces, 183
shortcuts, 180, 181
shortcuts and relationships, 181
business view for SAP BW metadata, 240
folders, 242
namespaces, 242
shortcuts, 240
shortcuts and dimensions, 242

C

cached data, 310
reusing, 315
cached metadata, 185
calculated aggregations
relational metadata, 143
calculated aggregation type, 185
calculated key figures, 191
calculated members
definition, 571
calculations
creating for relational metadata, 153
creating for SAP BW metadata, 231
security, 256
Calendar Type, 537
cardinality
1-1, 355
1-n, 355
checking, 338
data source, 81
definition, 571
dimensions and facts, 338
facts, 185
mixed, 185
notation, 81
queries, 82, 322
redefining, 83
rules, 82, 322
types, 82, 322
catalog, 319
categories
for verifying, 247
CCLOutofMemory Error, 386
changed features, 25, 26
changes in model not in report, 265
changing
metadata, 289
type of SQL, 109
changing packages
analyzing the effects, 292
characteristic
mapping to Framework Manager, 197
characteristic dimensions
externalizing, 558
characteristic structures, 195
checking
projects, 247
relationships, 79
checking in
projects, 285
checking out
projects, 285
choosing
query processing, 315
circular joins, 365
clearing
object security, 255
CLOB
in stored procedures, 381
Oracle, 381
Clock, 537
coalesce statement, 355

Cognos 8.1 or 8.2
upgrading from, 378
Cognos 8 models
importing, 67
Cognos 8 Planning and SAP BW data, 563
Cognos PowerCubes
as data sources, 48
publishing, 263
Cognos SQL, 111
collation sequences, 59
comments
adding to SQL, 108
Common Warehouse Metamodel, 297
exporting to, 386
comparing tables, 99
complex expressions
relationships, 84
Composite data sources
connecting to, 57
Composite stored procedures, 91
concepts, 321
conditional query subjects, 161, 237
configuring
CVS repository connections, 282
conformed dimension
definition, 571
conformed dimensions, 194, 347
creating, 99
multiple facts, 352, 357
removing, 207
conformed query subjects, 194
conformed star schema groups, 347
conforming dimensions, 205
connecting to multiple PowerCubes, 58
connections
repositories, 280
connections between
dimensions, 127, 207
query subjects, 99
content manager data source, 319
Context Explorer, 99, 127, 207
controlling
access, 252
SQL generation, 310
converting
measures into query items, 124, 126
model query subjects into data source query subjects, 108
query items into measures, 153
query subjects into dimensions, 107
regular dimensions, 131
copying
projects, 290
creating, 297
branches, 272
calculations for relational metadata, 153
calculations for SAP BW metadata, 231
CVS repository connections, 282
data source connections, 61
dimensions for relational metadata, 115, 116, 123, 124
dimensions for SAP BW metadata, 199
filters for relational metadata, 156
filters for SAP BW metadata, 233
folders for relational metadata, 183
folders for SAP BW metadata, 242
links, 279
measure dimensions, 346
measure dimensions for relational metadata, 124
measure folders for relational metadata, 185
namespaces, 78
namespaces for relational metadata, 183
namespaces for SAP BW metadata, 242
packages, 250
parameter maps for relational metadata, 161
parameter maps for SAP BW metadata, 237
projects, 33
prompts with query macros, 169
query item folders for relational metadata, 184
query sets for relational metadata, 99
query subjects for relational metadata, 88, 89, 91
query subjects for SAP BW metadata, 211
regular dimensions, 344
regular dimensions for relational metadata, 117, 127
relationships for relational metadata, 84
relationship shortcuts for relational metadata, 85
repositories, 280
segments, 278
star schema groups, 347
Visual SourceSafe database connections, 281
Visual SourceSafe repository connections, 281

creating local projects from repositories, 283
cross-fact queries, 338
cross-product joins, 310
crowsfeet notation, 38
csv files, 259, 261
CSVIdentityName function, 254
CSVIdentityNameList function, 255
cubes, 319
  definition, 571
curly brackets, 165
currency
  format type, 148, 219
Currency, 538
Currency Display, 538
Currency Symbol, 538
Currency Symbol Position, 538
custom relationship expressions, 83
CVS repository connections
  configuring, 282
  creating, 282
CWM, *See* Common Warehouse Metamodel

D

data
  formatting for relational metadata, 148
  formatting for SAP BW metadata, 219
  formatting using patterns, 545
  security, 252, 253
  using multilingual, 132
database connections, *See* data source connections
database functions
  vendor-specific, 317
database layer, 79
database only, 315
databases
  importing, 66
  sorting, 59
data integrity
  maintaining, 59
data source
  adding functions, 317
  updated by stored procedure, 91
data source connections, 61
  accessing, 61
  Cognos PowerCubes, 48
  Composite Software, 57
  creating, 61
  isolation levels, 59
  MS SQL, 61
  ODBC, 53
  OLAP cubes, 50
  testing, 64
  using network paths, 64
  XML, 57
data source functions
  adding, 317
data source query subjects
  creating, 88
  definition, 33
data source query subjects for relational metadata, 87
  determinants, 94, 96
  using parameters, 164
data sources
  Architect XML files, 68
  Cognos 8 models, 67
  connection levels, 59
  DB2 OLAP, 53, 54
  DecisionStream, 68
  definition, 572
  ERwin, 72
  Hyperion Essbase, 53
  IBM DB2 Cube Views, 72
  Impromptu XML files, 68
  Microsoft Analysis Server, 51
  Microsoft SQL Server, 50
  modifying properties, 319
  multilingual, 133
  Oracle Designer, 72
  properties, 202, 315
  relational databases, 66
  SAP BW, 191
  third-party metadata sources, 72
  XML files, 77
datatypes
  graphic, 383
data types, 373
data warehouse
  relationships, 79
date
  format type, 148, 219
  prompts, 150, 221
Date Ordering, 538

date prompts, 149, 220
Date Separator, 538
Date Style, 538
DB2, 384
DB2 Cube Views
  imported expressions, 77
DB2 OLAP
  data sources, 53, 54
Decimal Separator, 538
DecisionStream
  importing, 68
deconforming SAP BW dimensions, 207
default data set
  setting, 298
defining
  function sets, 317
  languages for relational metadata, 135
  prompt controls, 149
  prompt controls for SAP BW metadata, 220
deleting
  projects, 292
dense data
  semi-additive measures, 146
dependencies, object, 294
dependencies, report, 294
deprecated features, 25, 26
detailed fact query subject, 563
detailed key figures, 563
detecting
  relationships, 86
determinants, 94
  cardinality, 81
  converting from dimension information, 374
  defining, 324
  query subjects, 333
  relationships, 185
  specifying, 94, 96
  SQL generation, 98
  uniquely identified, 94, 96
diagrams, 35
  settings, 38
  viewing, 38
Diagram tab, 38
dimensional data, 343
dimensionally modeled relational metadata
  semi-additive measures, 146
dimensionally modeling relational metadata, 344
dimensional models
  star schemas, 176
dimensional queries, 351
  multiple facts and grains, 352, 357
  single fact, 351
dimension information, 371, 374
dimension map, 115, 116, 123, 124, 199
Dimension Map tab, 39
dimensions
  ambiguous, 361
  attributes of, 57
  conformed, 194
  converting from query subjects, 374
  creating, 39
  definition, 33, 572
  exploring, 38
  hierarchies, 345
  identifying, 338
  measure, 335, 346
  mixed cardinality, 185
  model, 327
  modifying, 39
  properties, 39
  query subjects, 327
  regular, 327, 333, 335, 344
  role-playing, 339
  SAP BW, 557
  searching, 41
  semi-additive measures, 146
  shared, 335
  star schema groups, 347
dimensions for relational metadata, 115, 176
  balanced hierarchies, 120
  converting, 131
  converting from query subjects, 107
  dimension map, 115, 116, 123, 124
  exploring, 127
  hierarchies, 119
  keys, 122
  levels, 119, 121
  measure, 124
  merging, 127
  modifying, 40
  network hierarchies, 121
  ragged hierarchies, 121

regular, 117
roles, 123, 124
scope relationships, 126, 127
shortcuts, 181
snowflaked, 179
testing, 128
unbalanced hierarchies, 121
dimensions for SAP BW metadata, 198
balanced hierarchies, 200
conforming, 205
deconforming, 207
dimension map, 199
hierarchies, 200
key figures, 205
levels, 200, 202
network hierarchies, 202
ragged hierarchies, 201
regular, 199
roles, 202, 204
shortcuts, 242
testing, 208
unbalanced hierarchies, 201
Display AM / PM Symbols, 539
Display As Exponent, 539
Display Days, 539
Display Eras, 539
Display Hours, 539
Display Milliseconds, 539
Display Minutes, 539
Display Months, 539
Display Seconds, 540
Display Time Zone, 540
Display Weekdays, 540
Display Years, 540
Divide By Zero Characters, 540
DMR (dimensionally modeled relational) metadata, 344
documenting the model, 268
double-counting, 94, 338, 355, 361
double quotation marks, 165
drop-down list prompts, 149, 150, 220, 221
duplicate object names
importing, 65

E

editing
SQL, 108
embedded files, 259, 261
embedded strings, 165
in expressions, 165
in macros, 165
environment
Framework Manager, 31
error messages
QE-DEF-0177, 383
QE-DEF-0259, 384
QE-DEF-0260 parsing error, 385
UDA-SQL-0107, 383
UDA-SQL-0114, 383
errors
repairing, 248
ERWin metadata
out of memory error, 382
escape symbol, 165
Essbase, *See* Hyperion Essbase
evaluating
relational query subjects, 106
SAP BW query subjects, 213
Event Studio
stored procedures, 91
examples for relational metadata
filter, 160
in_range function, 156
multilingual modeling, 137
parameter map, 162
prompts, 171
examples for SAP BW metadata
in_range function, 233
except all query set, 99
except query set, 99
Explorer tab, 38
exploring
dimensions, 115, 127, 199, 207
object-based security, 267
objects, 38
packages, 267
query subjects, 99
roles, 267
Exponent Symbol, 540
exporting
metadata as CWM files, 297
translation tables, 136

Exporting
to a CWM file, 386
expression editor
Block Functions, 419
Business Date/Time Functions, 414
Common Functions, 427
Constants, 411
Constructs, 413
Data type casting functions, 532
DB2, 446
DB2 Cast, 458
DB2 Math, 459
DB2 Trigonometry, 460
function sets, 317
Informix, 462
Informix Math, 466
Informix Trigonometry, 467
Macro Functions, 420
Member Summaries, 408
MS Access, 468
MS Access Cast, 474
MS Access Math, 475
MS Access Trigonometry, 475
Operators, 389
Oracle, 476
Oracle Math, 483
Oracle Trigonometry, 483
Red Brick, 485
Report Functions, 516
SAP BW, 506
SAP BW Math, 507
SAP BW Trigonometry, 506
searching for values, 388
SQL Server, 491
SQL Server Cast, 496
SQL Server Math, 497
SQL Server Trigonometry, 498
Summaries, 396
Sybase, 507
Sybase Math, 514
Sybase Trigonometry, 514
Teradata, 499
Teradata Trigonometry, 503
expressions
braces, 384
custom, 83
errors, 384
imported from third-party metadata, 77
relationships, 84
using prompts, 149
externalized dimensions
prompts, 385
externalize methods
IQD, 27
externalizing
characteristic dimensions, 558
Framework Manager requirements, 560
InfoCube key figures, 558
measure dimensions, 558
model query subjects, 558
query subjects, 259, 261
SAP BW Dimensions, 557
SAP BW Query key figures, 558, 560
SAP BW Query requirements, 559

F

fact data, 343
fact-less query, 347
facts, 346
aggregation rules, 146
ambiguous, 361
cardinality, 185
identifying, 338
mixed cardinality, 185
fact tables, 124, 176
features
changed, 25, 26
deprecated, 25, 26
new, 25
removed, 25, 27
file-based data sources
recommendation, 64
filters
security, 253, 256
filters for relational metadata
applying to query subjects, 158
creating, 156
determining usage, 158
mandatory, 156
modifying, 156
multiple dimensions, 40
multiple query items, 40

multiple query subjects, 40
optional, 156
filters for SAP BW metadata
applying, 235
creating, 233
determining usage, 235
modifying, 233
finding
objects, 41
finding information, 24
fixing
failed transactions, 275
folders
creating, 183, 185
measure, 185
query item, 184
using shortcuts, 180
folders for SAP BW metadata
creating, 242
using shortcuts, 240
font
settings, 38
foreign keys, 86
format strings, 54
formatting
data for relational metadata, 148
data for SAP BW metadata, 219
data using patterns, 545
format types
currency, 148, 219
date, 148, 219
text, 148, 219
Framework Manager
project, 31
security filters change for SAP BW, 26
full outer joins, 81, 355, 383
functions
errors, 384
project, 317
function set ID, 319
function sets
adding, 250, 317
defining, 317

G

gateway
cannot access URI, 382
generating
relationships, 86
Global Customer Services Web site, 24
governors
allow usage of local cache, 315
Cognos 8 upgrade, 379
Cognos ReportNet upgrade, 372
definition, 572
setting, 310
granularity, 342
graphic datatype, 383
groups
adding, 253
star schema, 176
Group Size (digits), 540

H

help
getting, 24
hierarchies, 324, 327, 361, 374
balanced, 120, 200, 244
DB2 OLAP, 54
definition, 572
multiple, 345
network, 121, 202
ragged, 121, 201, 244
regular dimension, 119, 200
SAP BW, 193
specifying roles, 123, 124
unbalanced, 121, 201, 245
versioned, 193
history
viewing, 286
Hyperion Essbase
data sources, 53

I

IBM DB2, *See* DB2
identifiers
unique, 94
imported metadata
checking, 338

importing
Architect XML files, 68
Cognos 8 models, 67
databases, 66
DecisionStream, 68
duplicate object names, 65
Impromptu XML files, 68
metadata from XML files, 77
metadata into Framework Manager, 65
relational metadata, 86
relationships, 79
SAP BW hierarchies, 193
SAP BW metadata, 191
SQL View definitions, 76
third-party data sources, 72
translation tables for relational metadata, 137
import view, 79
Impromptu Query Definition files, 259, 261
Impromptu XML files
importing, 68
improving performance
reusing cached data, 315
in_range function, 156, 233
InfoCube
mapping to Framework Manager, 197
permissions for accessing metadata, 192
InfoCube key figures
externalizing, 558
information
finding, 24
Informix
functions, 91
stored procedures, 91
inner joins, 81, 86, 386
International Currency Symbol, 540
intersect all query set, 99
intersect query set, 99
IQD
files, 259, 261
IQD externalize method, 27
isolation levels, 59

J

Japanese
characters in export path, 386
joins, 79
circular, 365
full outer, 81, 355, 383
inner, 81, 86, 386
loop, 341
modifying, 83
outer, 81, 83

K

key figures, 191
mapping to Framework Manager, 197
modifying, 205
not added after importing, 381
prompts, 385
structures, 195
keys, 116, 199, 374
cardinality, 81
foreign, 86
for levels, 122
for roles, 202, 204
primary, 86
specify roles, 123, 124

L

languages
adding for relational metadata, 135
defining for relational metadata, 135
incorrect in SAP BW query, 383
setting up multilingual environment, 132
specifying for packages, 258
layers, 79
level of details
settings, 38
levels, 115, 119, 199, 200, 202, 204, 361, 374
definition, 572
member unique names, 122, 202
regular dimension, 121, 202
security, 385
specify roles, 123, 124
unique, 117
limitations
linking, 277
macros, 168
segmenting, 277
limited local, 315

linked projects, 276, 380
synchronizing, 307
linking
limitations, 277
links, 276
creating, 279
literal strings, 165
in expressions, 165
in macros, 165
locales
definition, 572
Microsoft Analysis Server, 51
locales, *See Also* languages
local query processing
rollup processing, 314
log files, 300
play back, 301
logging on
to MS SQL, 61
loop joins, 338, 341

M

macros
creating, 168
limitations, 168
prompts, 165
using, 165
main projects, 278
maintaining
data integrity, 59
managing
packages, 255, 258, 267, 268
project, 267
projects, 290, 300
mandatory cardinality, 82
mandatory prompts, 169
Mantissa (digits), 541
many-to-many relationships, 81
many-to-one relationships, 80
mapping
parameter values for relational metadata, 161
parameter values for SAP BW metadata, 237
SAP BW objects to Framework Manager, 197
master-detail tables, 343, 346
maximum cardinality, 82, 322
Maximum No. of Digits, 541
measure dimensions, 335
creating, 346
externalizing from SAP BW, 558
role-playing, 339
measure dimensions for relational metadata
creating, 124
exploring, 127
scope relationships, 126, 127
measure dimensions for SAP BW metadata
exploring, 207
measure folders, 185
measures
calculated aggregation type, 185
converting, 126, 153
converting to query items, 124
definition, 572
for Cognos 8 Planning, 563
format strings, 54
not added after importing, 381
semi-additive, 146
measure scope issues, 126
members
definition, 573
member unique names
relational metadata, 122
SAP BW metadata, 202
merging
projects, 269
query subjects, relational, 98
query subjects, SAP BW, 212
regular dimensions, relational, 127
merise notation, 38
metadata
cached, 185
changing, 289
exporting, 297
importing, 65
merging, 98, 212
moving, 289
publishing, 247
relational, 79
SAP BW, 191
security, 252, 258
synchronizing, 306
understanding, 267

methodologies
branching, 270
Microsoft
SQL server and logon, 61
SQL Server data sources, 50
Microsoft Analysis Server
data sources, 51
single signon, 51
Microsoft SQL Server
single signon, 50
migrating
models, 299
Minimized SQL, 114, 185
model query subjects, 115
minimum cardinality, 82, 322
Minimum No. of Digits, 541
Missing Value Characters, 541
model
versioning, 265
model advisor, 185
model business views, 175, 240
model documentation, 268
modeling
data sources, 133
distributed models, 269
main projects, 269
multilingual, 133
multiuser, 269
modeling problems, 185
model objects, 35
shortcuts, 333
using parameters, 161, 237
viewing, 35
model portability, 299, 310
model query subjects
definition, 33
externalizing, 558
security, 256
model query subjects for relational metadata, 89
creating, 89
creating from existing objects, 98
determinants, 94, 96
SQL type, 115
model query subjects for SAP BW metadata, 211
creating, 211
creating from existing objects, 212
models, 31
analyzing, 185
definition, 32, 573
migrating from one relational database to another, 299
preparing relational, 79
preparing SAP BW metadata, 191
publishing, 247
sample, 44
upgrading, 35, 371
upgrading Cognos 8, 378
upgrading Cognos ReportNet, 371
verifying, 247
versioning, 261
model segments
definition, 573
model session parameters
relational metadata, 163
SAP BW metadata, 238
model versioning, 265
model versions
updating, 261
modifying
conformed dimension, 207
filters for relational metadata, 156
filters for SAP BW metadata, 233
package access, 258
package administrative access, 258
packages, 250
properties, 39
query item properties, 40
query items for relational metadata, 139
query items for SAP BW metadata, 213
relational dimensions, 40
relational query subjects, 40
relationships, 83
session parameters, 163, 238
SQL at runtime, 169
stored procedure query subjects, 91
moving
metadata, 289
projects, 291
MultiCube
mapping to Framework Manager, 197
multidimensional
query subjects, 211

multilingual
data, using, 132
data sources, 133
multilingual modeling
example for relational metadata, 137
packages, 258
parameters, 134
relational metadata, 131
multilingual reporting environment
setting up, 132
multiple cubes in packages, 263
multiple data sets, 298
setting default data set, 298
multiple data source connections
using, 58, 61
multiple-fact queries, 327, 351, 352, 357
multiple-grain queries, 327, 352, 357
multiple hierarchies, 345
multiple relationships, 185
multiple users
modeling, 269
multiple valid relationships, 339, 341
multiuser modeling, 269

N

namespaces, 31
creating, 78
creating for relational metadata, 183
creating for SAP BW metadata, 242
definition, 32, 573
naming conventions
objects, 43
native metadata, 50
native SQL, 112
Negative Pattern, 541
Negative Sign Position, 541
Negative Sign Symbol, 541
nested parameters, 161, 237
network hierarchies, 121, 202
new features, 25
No. of Decimal Places, 541
non-additive, 146, 218
non-conformed dimensions, 357
normalization
definition, 573
normalized data sources, 343
notation, 38, 80, 81
null values
SAP BW, 310
number sign
reserved for macros, 165
numeric formats, 54
Numeric Overflow Characters, 542

O

objects
exploring, 38
naming conventions, 43
properties, 39
remapping, 295
searching for, 41
unique identifiers, 43
object security, 252
adding, 255
calculations, 256
exploring, 267
filters, 256
model query subjects, 256
removing, 255
shortcuts, 256
ODBC
data source, 53
OLAP cubes
as datasources, 50
OLAP data sources
publishing, 263
one-to-many relationships, 80, 355
one-to-one relationships, 80, 355
opening
projects, 35
optimizing PowerCubes, 48
optional cardinality, 82, 322
optional prompts, 169
optional relationships, 81
Oracle
Designer, 76
Oracle full outer joins, 383
organizing models, 175, 240
outer joins, 81, 86, 310
Analysis Studio, 83
full, 81, 355, 383
out of memory error, 382

P

package administrative access
    modifying, 258
packages, 31
    access, 258
    adding security, 258
    administrative access, 258
    analyzing changes, 292
    containing multiple cubes, 263
    creating, 250
    definition, 32, 573
    exploring, 267
    large, 386
    managing, 255, 258, 267, 268
    modifying, 250
    multilingual support, 258
    publishing, 247, 261
    security, 252
    verifying, 247
    viewing inclusion, 268
Padding Character, 542
parameter maps for relational metadata
    creating, 161
    using as lookup tables, 131
parameter maps for SAP BW metadata, 237
    creating, 237
parameters
    in data source query subjects, 164
parameters for relational metadata
    aliases, 161
    creating, 161
    data source query subjects, 164
    model objects, 161
    multilingual modeling, 134
    nested, 161
    session, 163
    using, 165
parameters for SAP BW metadata
    creating, 237
    maps, 237
    session, 238
pass-through SQL, 113
Pattern, 542
patterns
    using to format data, 545
pcoptimizer, 48
Percentage Symbol, 542
Percent Scale (integer), 542
performance, optimizing for SAP BW, 243
performance tuning
    query processing, 315
physical layer, 79
play back
    action logs, 302
plus operator, 165
portability, 299
PowerCubes
    building from SAP BW data, 561
    connecting to multiple data sources, 58
    Framework Manager guidelines, 562
    optimizing, 48
    Transformer guidelines, 562
presentation hierarchy level
    mapping to Framework Manager, 197
presentation layer, 79
primary keys, 86
project
    managing, 267
project objects
    naming conventions, 43
project page, 35
projects, 31
    adding to a repository, 283
    adding to repositories, 283
    branching, 269
    checking in, 285
    checking out, 285
    copying, 290
    creating, 33
    creating from repositories, 284
    definition, 31, 573
    deleting, 292
    files, 31
    fixing synchronization errors, 305
    Framework Manager, 31
    functions, 317
    linking, 279
    managing, 290, 300
    merging, 269
    moving, 291
    opening, 35

play back transactions, 301
renaming, 291
segmenting, 278
shortcuts, 279
synchronizing, 306
version control, 280
versions, 286
view history, 283
view transaction history, 301
Project Viewer, 35
Prompt Info property, 149, 220
prompts
cascade on reference, 151, 222
creating, 169
creating with macros, 169
date, 149, 150, 220, 221
defining, 149, 220
different values to use and display, 152, 222
display item reference, 152, 222
drop-down list, 149, 150, 220, 221
examples, 171
filter on reference, 152, 223
in expressions, 149
key figures, 385
mandatory, 169
optional, 169
properties, 149, 150, 220, 221
SAP BW variables, 223
search, 149, 150, 220, 221
stored procedure example, 94
testing, 152, 223
text box, 150, 220, 221
time, 149, 150, 220, 221
type-in, 149, 150, 220, 221
use item reference, 152, 222
properties
aggregation rules, 141
aggregation rules for SAP BW metadata, 216
cascade on reference, 151, 222
catalog, 319
content manager data source, 319
cubes, 319
data sources, 315
display item reference, 152, 222
filter on reference, 152, 223
function set ID, 319
modifying, 39
parameter map, 319
prompt info, 149, 220
prompt type, 150, 221
query interface, 319
query processing, 319
query type, 319
regular aggregate, 141, 216
rollup processing, 319
SAP BW data source, 202
SAP BW variables, 202
schema, 319
semi-aggregate, 216
usage, 141, 216
use item reference, 152, 222
viewing, 39
publishing
definition, 574
metadata, 247
packages, 247, 261, 265
packages based on OLAP data sources, 263

Q

QE-DEF-0177, 383
QE-DEF-0259, 384
QE-DEF-0260 parsing error, 385
queries
fact-less, 347
multiple-fact, 327, 351, 352, 357
multiple-grain, 327, 352, 357
single fact, 351
split, 362, 365
stitched, 338
Query, SAP BW
mapping to Framework Manager, 197
query items
converting, 126, 153
converting from measures, 124
definition, 33, 574
properties, 39
shortcuts, 386
query items for relational metadata
aggregation rules, 146
folders, 184
modifying, 40, 139
roles, 123, 124

usage, 145
query items for SAP BW metadata
aggregation, 218
aggregation rules, 218
formatting, 219
modifying, 213
roles, 202, 204
usage, 217
query macros
using to create prompts, 169
query processing, 315, 319
choosing, 315
query reuse, 315
query sets for relational metadata
creating, 99
testing, 103
Query Studio
QE-DEF-0260 parsing error, 385
query subjects
conforming, 194
converting to dimensions, 374
csv files, 259, 261
DB2, 384
definition, 33, 574
determinants, 185, 333, 374
dimensions, 185, 327
embedded files, 259, 261
exploring, 38
externalized, 259, 261
facts, 185
hierarchies, 361
Impromptu Query Definition files, 259, 261
minimized SQL, 185
mixed cardinality, 185
multiple relationships, 185
properties, 39
SAP BW, 373
searching, 41
security, 253
star schema groups, 347
tab files, 259, 261
testing, 76
third party, 76
Transformer files, 259, 261
using in Transformer, 259
query subjects for relational metadata
changing type of SQL, 109
converting, 107, 108
converting from dimensions, 131
creating, 88, 89, 98
data source, 87
determinants, 94, 96
editing SQL, 108
evaluating, 106
exploring, 99
generating SQL, 114
model, 89
modifying, 40, 91
parameter maps, 161
query sets, 99
relationships, 79
shortcuts, 181
SQL, 87
stored procedure, 90
testing, 103
types, 87
updating, 107
query subjects for SAP BW metadata
creating, 211, 212
evaluating, 213
model, 211
parameter maps, 237
query type, 319
quick tours
using, 24
quotation marks, 165
in expressions, 165
in macros, 165

R

ragged hierarchies, 121, 201, 244
recommendations
using network paths for file-based data sources, 64
recording transactions, 300
recursive relationships, 185, 342
reflexive relationships, 185, 342
regional settings
setting up a multilingual environment, 132
Regular Aggregate property, 146
relational metadata, 141
SAP BW metadata, 216, 218

regular dimensions, 327, 333, 335
creating, 344
hierarchies, 345
role-playing, 339
regular dimensions for relational metadata, 117
converting, 131
exploring, 127
hierarchies, 119
levels, 119, 121
merging, 127
scope relationships, 126, 127
regular dimensions for SAP BW metadata, 199
exploring, 207
hierarchies, 200
levels, 200, 202
related documentation, 23
related objects, 99, 127, 207
relational data source query subjects
evaluating, 106
updating, 107
relational metadata, 79
business rules, 153
business view, 79
import view, 79
query items, 139
star schemas, 176
relational modeling concepts, 322
relational models
migrating, 299
relationships
0-1, 80
0-n, 80
1-1, 80
1-n, 80, 355
ambiguous, 338
cardinality, 80, 322
changing, 83
checking, 79, 338
complex expressions, 84
creating, 84
custom, 83
definition, 79
detecting, 86
determinants, 185
from Oracle Designer, 76
generating, 86
importing, 79
levels of granularity, 342
many-to-many, 80, 81
many-to-one, 80
modifying, 83
multiple, 185
multiple valid, 339, 341
n-n, 80
one-to-many, 80
one-to-one, 80
reflexive and recursive, 185
scope, 126, 127
shortcuts, 85, 181
verifying, 79
remap objects, 295
RemoteCube
mapping to Framework Manager, 197
removed features, 25, 27
removing
object security, 255
renaming
projects, 291
relationships, 83
repairing
Cognos ReportNet models, 374
repairing models, 248
report
dependencies, 294
reporting requirements, 29
ReportNet
upgrading from, 371
reports
not showing changes to model, 265
repositories
adding projects, 283
branching, 289
connecting to, 280
creating, 280
creating projects based on, 284
get latest version of a project, 283
latest version of project, 283
segmented projects, 287
synchronizing projects, 308
using, 280, 283
repository control, 280
adding projects, 283

reserved symbols
number sign, 165
reserved words, 565
resetting
Regular Aggregate property, 147
Usage property, 147
resolving
ambiguous objects, 361
split queries, 362, 365
restricting
access, 252
BLOBs, 310
data retrieved, 310
result set of stored procedure, 381
reusing cached data, 315
role-playing dimensions, 185, 339
roles
adding, 253
dimensions, 123, 124
exploring, 267
roles for SAP BW metadata
dimensions, 202, 204
rollup processing, 314, 319
row limits, 310
rules
aggregation for SAP BW metadata, 216
business, 153
business for SAP BW metadata, 231
cardinality, 82
rules of cardinality, 322

S

sample models, 44
SAP BW
importing key figures, 191
security filter change in Framework Manager, 26
Transformer, 557
wrong language in query, 383
SAP BW data
Cognos Planning, 563
SAP BW dimensions, 557
SAP BW metadata, 191
importing, 191
SAP BW Query
permissions for accessing metadata, 192
SAP BW Query key figures
externalizing, 558
SAP BW query subjects, 373
SAP BW structures, 195
SAP BW variables, 202, 223, 245
save as
copying projects, 290
Scale, 542
schema, 319
scope relationships, 115, 116, 123, 124, 126, 127, 199
script files
fixing errors, 305
scripts
running, 265
searching
objects, 40, 41
searching for values
unexpected results, 388
search prompts, 149, 150, 220, 221
Secondary Group Size (digits), 542
secured InfoCube
accessing, 191
security
CSVIdentityName function, 254
CSVIdentityNameList function, 255
exploring, 267
filter change in Framework Manager for SAP BW, 26
filters, 253
levels, 385
packages, 250
types, 252
Security Error Characters, 542
segmented projects, 276, 278, 380
copying, 290
deleting, 292
in repositories, 287
moving, 291
synchronizing, 307
segmenting
limitations, 277
projects, 278
segments, 276
recommendations for using, 276
self-joins, 185
semi-additive, 147

semi-additive measures, 146, 219
Semi-Aggregate property, 147
SAP BW metadata, 216, 219
session parameters
creating, 163
modifying, 163, 238
setting
governors, 310
settings
font, 38
level of detail, 38
notation, 38
snap options, 38
test, 104, 130, 209
shared dimensions, 335
shortcuts, 333, 386
behavior, 310, 311
security, 256
shortcuts for relational metadata
dimensions and query subjects, 181
relationships, 85, 181
using, 180
shortcuts for SAP BW metadata
dimensions, 242
using, 240
show dependencies, 294
single fact queries, 351
single quotation marks, 165
single signon
Microsoft Analysis Server, 51
Microsoft SQL Server, 50
snap options
settings, 38
snowflaked data sources, 343
snowflaked dimensions, 179
sorting
databases, 59
objects, 40
sparse data
semi-additive measures, 146
sparse data and Analysis Studio, 83
split queries, 362, 365
SQL, 351
1-1, 357
1-n, 357
As View, 114, 115
changing type, 109
Cognos, 108, 111
comments, 108
determinants, 98
editing, 108
generation types, 114
importing from Power Designer file, 76
Minimized, 114, 115, 185
model query subjects, 115
multiple-fact multiple-grain queries, 359
native, 108, 112
pass-through, 108, 113
query subjects, 87
server, connecting, 61
split queries, 363, 367
stand-alone native, 108, 113
stitched queries, 354
SQL for inner joins, 386
SQL generation
controlling, 310
square brackets, 165
star dimensions, 179
star schema concepts, 342
star schema groups, 335
creating, 347
multiple conformed, 347
single fact query, 351
star schemas
groupings, 176
multiple, 178
statistics
viewing, 41
stitched queries, 338
stored procedure query subjects, 90
Composite, 91
creating, 91
definition, 33
evaluating, 106
example, 94
Informix, 91
modifying, 91
updating, 107
stored procedures
data query updates database, 91
unable to view, 381

structures
SAP BW, 195
summaries
viewing, 41
summarize aggregations
relational metadata, 145
supported environments
multilingual, 131
supporting
multiple users, 269
synchronizing
fixing errors, 305
linked projects, 307
metadata, 306
projects, 306
query subjects, 107
segmented projects, 307
syntax
BmtScriptPlayer, 303
errors, 384

T

tab files, 259, 261
tasks
viewing, 41
testing
calculations, 382
changing settings, 104, 130, 209
data source connections, 64
projects, 247
prompts, 152, 223
query subjects, 76, 383
relational measure dimensions, 128
relational query sets, 103
relational query subjects, 103
relational regular dimensions, 128
SAP BW measure dimensions, 208
SAP BW regular dimensions, 208
text box prompts, 149, 150, 220, 221
text format type, 148, 219
Thousands Separator, 543
time limits, 310
time prompts, 149, 150, 220, 221
Time Separator, 543
Time Style, 543
Time Unit, 543
trace
transaction history, 301
transaction history
viewing, 301
transactions
fixing, 275
play back, 301
Transformer
SAP BW, 557
Transformer 7.x, 557
Transformer 8.x, 557
Transformer and query subjects, 259, 261
translation tables
exporting, 136
importing, 137
troubleshooting, 381
type-in prompts, 149, 150, 220, 221
type-in SQL, 384

U

UDA-SQL-0107, 383
UDA-SQL-0114, 383
unable to access service at URL, 382
unbalanced hierarchies, 121, 201, 245
DB2 OLAP, 54
understanding
metadata, 267
union all query set, 99
union query set, 99
unique identifiers, 94
objects, 43
unique keys, 374
unique levels, 117
uniquely identified determinants, 94, 96
UNIX
creating ODBC data sources, 53
updating
model versions, 261
query subjects, 107
upgrading
Cognos 8 models, 378, 379
Cognos ReportNet models, 371, 372
linked projects, 380
models, 35, 371
segmented projects, 380

upgrading models
key figures, 381
warnings, 377
URI
unable to access service, 382
Usage property
relational metadata, 141, 145
SAP BW metadata, 216, 217
users
adding, 253
packages, 250
Use Thousands Separator, 543
using
multiple data sets, 298
multiple data source connections, 58, 61
using embedded strings
in expressions, 238
in macros, 238
using literal strings
in expressions, 238
in macros, 238
using quotation marks
in expressions, 238
in macros, 238

V

validating
relational query subjects, 106
SAP BW query subjects, 213
valid relationships
multiple, 339
vargraphic datatype, 383
variables
SAP BW, 223, 245
stored procedure, 94
using when externalizing, 560
vendor-specific database functions, 317
verifying
Cognos 8 models, 379
Cognos ReportNet models, 371, 374
models, 247
packages, 247
relationships, 79
upgraded Cognos 8 models, 379
version control, 280
versioned hierarchies, 193
versioning
models, 261
projects, 286
view history
projects in repositories, 283
viewing
diagrams, 38
history, 286
model objects, 35
package inclusion, 268
properties, 39
related objects, 99, 127, 207
SQL, 108
statistics, 41
summaries, 41
tasks, 41
views from PowerDesigner, 76
Visual SourceSafe database
creating connections, 281
Visual SourceSafe repository connections
creating, 281

W

With clause, 310

X

XML data sources
connecting to, 57
XML files
importing, 77

Z

Zero Value Characters, 543